AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING, AMARAVATI

Department of Computer Science and Engineering

MINI - PROJECT REPORT ON

BIG DATA-DRIVEN GRID STABILITY PREDICTION USING PYSPARK AND MACHINE LEARNING

Submitted

By

K Balavignesh Reddy (AV.EN.U4AIE22016) **P Gagan Devesh (AV.EN.UAIE22027)**
Kabin Dev (AV.EN.UAIE22015) **D Sai Parthava Naidu (AV.EN.U4IE22010)**

B. Tech (AIE) SIXTH SEMESTER

During the academic year

2024- 2025

Under the Guidance of

Dr. V LAKSHMI CHETANA

Assistant Professor (Sl. Gr.)

Department of Computer Science and Engineering

Amrita School of Computing

Amrita Vishwa Vidyapeetham, Amaravati Campus

Contents

ABSTRACT

With the era of power distribution having become modern today, the reliability of power grids has become a serious issue now. Growing quantities of power being fed from sources of renewables onto the grid, variable patterns of demands, and random system disturbance are causing genuine predictability within grid stability a top priority today. Fixed threshold rule-based systems traditionally being implemented for monitoring the grid can't identify sophisticated patterns causing instability.
This paper introduces Big Data analytics with PySpark data-driven Grid Stability Prediction. Big-dimensional power system data are taken into account in this paper, where Machine Learning (ML) algorithms like Support Vector Machines (SVM), Random Forest (RF), Multi-Layer Perceptron (MLP), and Deep Learning (DL) algorithms are used in the stability evaluation. A few of the real-time grid quantities are included in data utilized for training, namely, voltage fluctuations, frequency excursions, power factor swings, and load distribution patterns.
As the smart grid produces a vast amount of data, we use PySpark for effective preprocessing, feature extraction, and model training. Using cross-validation methods, we optimize model hyperparameters to ensure optimal performance. Experimental results show that SVM performs better than other models with a remarkable accuracy of 99.86%, followed by MLP and Deep Learning models. Comparative analysis shows the strength of different algorithms in dealing with different stability scenarios.
Aside from performance analysis, the paper describes essential issues such as data imbalance, computational complexity, and applicability to real-time prediction. The results provide a solid basis for deploying stability monitoring systems of AI-based for power grids. Future research can investigate deep reinforcement learning and ensemble models in an attempt to further improve predictive performance.
This work contributes to power system analysis as it provides a novel solution for real-time grid stability assessment with the potential to allow energy suppliers to avoid blackouts and provide an uninterrupted supply of electricity.

Keywords: Grid Stability Prediction, Big Data Analytics, Machine Learning, PySpark, Smart Grids, Real-time Monitoring.

1. INTRODUCTION

The reliability of power grids for electricity is a key consideration in providing timely, uninterrupted, and efficient electricity supply to industries, residences, and businesses. As the energy consumption of the world keeps increasing, power grids are becoming more advanced as renewable energy sources are integrated, energy usage increases, and decentralized power generation technologies are in place. These conditions bring high variability and uncertainty in grid operations, and stability is a difficult task to maintain. Instability in a power grid can cause widespread blackouts, equipment failure, economic losses, and inefficiencies in operations, having a great impact on industries and daily life. Accessibility of power grids is one of the strongest influences on the sustenance of timely, consistent, and effective supply of electricity to industry, residential buildings, and trade. With ongoing global energy consumption on the rise, power grids are getting increasingly complicated with the inclusion of renewable sources of power, augmented use of energy, and decentralized power generation equipment.These conditions bring a great deal of uncertainty and variability to grid operations, and stability is an elusive condition to achieve. Instability in the power grid can cause massive blackouts, equipment failure, loss of revenue, and inefficiencies in operation, with significant effects on industry and everyday life.

Traditional grid stability analysis was based on rule-based models and deterministic methods, where power flow, frequency, and voltage oscillations were addressed by pre-defined human knowledge and equations. Although these approaches are very well appropriate for static setups, they experience issues with actual-time volatility, IoT device high-dimensional data sets, cyclic renewable energy supply volatilities, and time-varying load profiles. As power grids evolve, traditional methods lack strength in terms of real-time predictive analytics and forecasted decision-making. To overcome these limitations, new power systems have increasingly been heading towards data-centric practices with Big Data, Artificial Intelligence (AI), and Machine Learning (ML) to augment grid stability monitoring, analysis, and prediction.

The advancement in Big Data analytics and ML algorithms has made it possible to create smart grid stability assessment models that handle massive volumes of historical and real-time grid data, look for patterns, and provide predictive insights to the operators so that they can make decisions ahead of time before instability. This research focuses on designing a PySpark-based grid stability prediction system, which is a distributed computing environment facilitating the efficient processing of large amounts of data with parallel processing. With the high-performance computing capabilities of PySpark, this research ensures that the process is performed promptly in real time, which is very much suited for analyzing large amounts of power grid data from different sources.
To achieve this, the research uses different machine learning models for predicting and analyzing grid stability, such as:
Support Vector Machines (SVM): A stable classification framework that works effectively with high-dimensional data.
Random Forest Classifier: An ensemble classifier model that enhances prediction performance through multiple decision trees.
Multi-Layer Perceptron based method capable of learning grid stability information in complex patterns.

ANN is fully connected multilayered neural network trained for classification and feature extraction. They are trained and cross-checked against actual grid data, and their performance is tested based on significant performance metrics like accuracy, F1-score, precision, and recall. The work of this paper is to implement a scalable as well as a precise prediction platform for classifying stable and unstable grid conditions. Through the synergistic combination of feature engineering, class imbalance remediation (SMOTE-ENN), and hyperparameter optimization*, the model robustness and the predictive accuracy improve.

Through the integration of Big Data, Machine Learning, and Distributed Computing, this research will revolutionize grid monitoring, fault detection, and stability evaluation. The outcome will result in a stronger, more efficient, and intelligent power infrastructure, enabling better decision-making and preventive actions in smart power grids.

2. Problem Statement

Grid stability is the backbone of contemporary power grids, ensuring uninterrupted supply of electricity to industries, enterprises, and homes. With increasing use of variable energy sources, renewable energy sources, and sophisticated grid systems, stability is a formidable challenge. Grid instability has devastating outcomes, such as voltage collapses, frequency undulations, and extensive blackouts, halting economic processes and vital services.

Legacy grid stability analysis is extremely model-based and rule-based. While such techniques were acceptable in the earlier decades, they are not very capable of handling the enormous amounts of real-time data that modern power systems generate. Besides, they lack the flexibility necessary to identify and respond to stability threats in real-time. These limitations render data-driven approaches unavoidable, which are capable of processing big data, detecting patterns of instability, and providing grid operators timely forecasts.

New advances in Big Data and machine learning provide a solution to this issue that is promising. With distributed computing platforms like PySpark, one can efficiently handle large grid data sets and train real-time predictive models that can identify risks of instability. Despite these developments, there is still a need for large-scale comparative research of diverse machine learning models as prediction tools of grid stability. The central challenge lies in selecting the optimal model that strikes a balance between accuracy, computational expense, and scalability when handling high-dimensional energy data.

The objective of this study is to bridge this gap by systematic comparison of a number of machine learning algorithms like SVM, Random Forest, MLP, and Deep Learning on a large data set by PySpark. The performance of each will be compared in accuracy, precision, recall, and F1-score and will provide important insights into the most suitable approach for real-time grid stability prediction.

3. Objectives

The primary goal of this research is to develop a **Big Data-driven machine learning framework** for accurate and scalable **grid stability prediction** using **PySpark**. The study focuses on leveraging distributed computing and advanced machine learning models to handle large-scale energy datasets efficiently. The key objectives of this research are:

- **Develop an Efficient Grid Stability Prediction Model:**

Implement and optimize machine learning algorithms, including Support Vector Machines (SVM), Random Forest, Multi-Layer Perceptron (MLP), and Deep Learning (Keras-based networks), to accurately predict grid stability status. Design a high-performance predictive pipeline capable of handling large-scale, real-time grid data.

- **Enhance Model Generalization Using Cross-Validation :**

Apply k-fold cross-validation and hyperparameter tuning techniques to ensure model robustness across diverse grid scenarios and varying energy loads. Develop strategies to minimize overfitting and improve model adaptability to unseen data.

- **Compare the Performance of Various Machine Learning Models :**

Conduct an extensive comparative analysis of different machine learning models based on accuracy, precision, recall, and F1-score, ensuring a data-driven approach in selecting the most effective model. Evaluate computational efficiency, scalability, and real-time applicability of each model in a practical grid monitoring environment.

- **Bridge the Gap Between Traditional and AI-Driven Grid Monitoring:**

Identify the limitations of conventional deterministic methods in predicting grid stability and demonstrate how machine learning models offer more flexible and adaptive solutions.Integrate domain knowledge from power systems with AI-driven decision-making to provide reliable stability forecasts.

- **Provide Scalable Solutions for Grid Operators:**

Develop a real-world deployment framework that can be integrated into existing power grid monitoring systems.Offer recommendations for policymakers and energy providers on how machine learning-based grid stability prediction can enhance energy security and resilience.

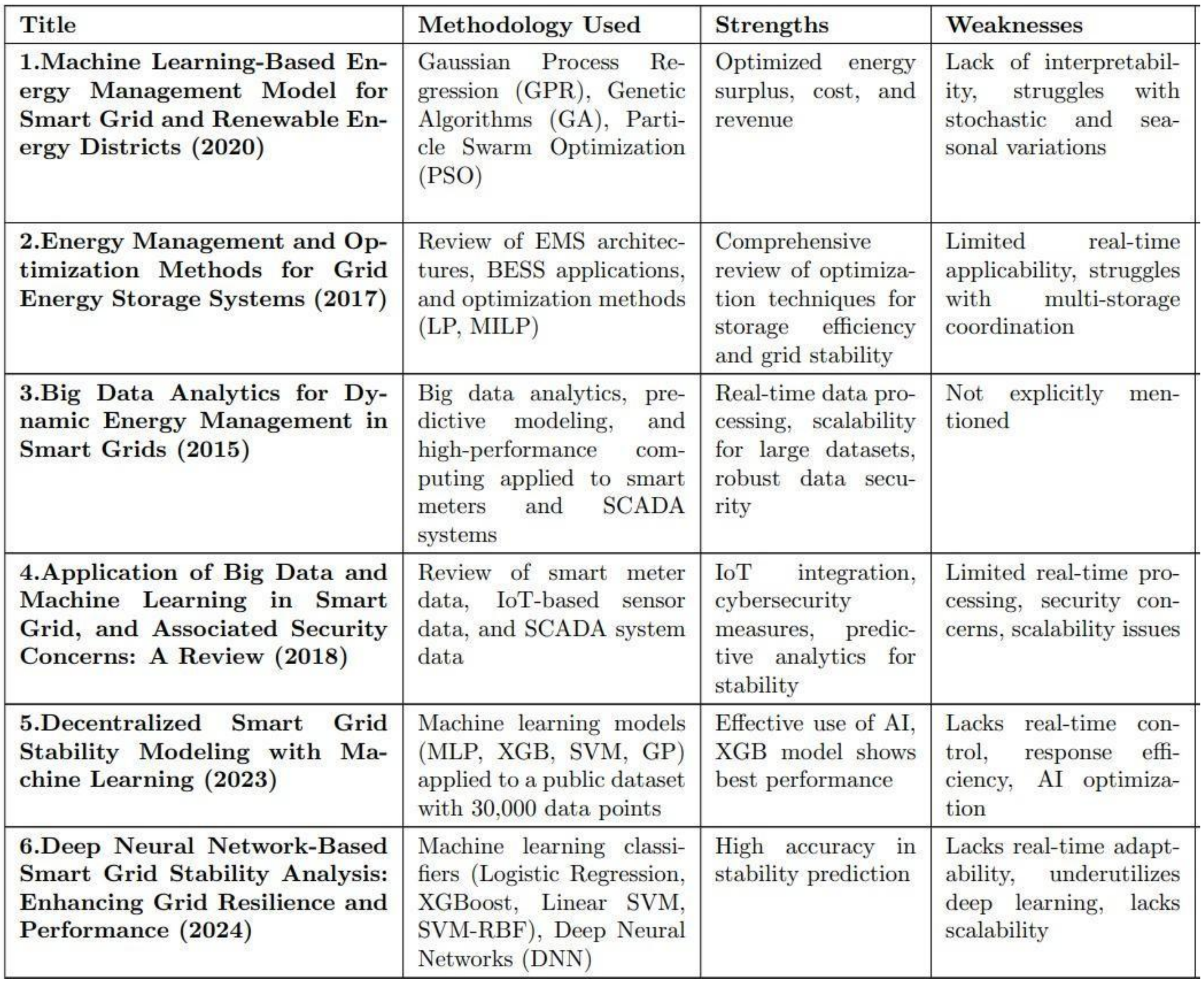
- **Explore Future Directions for Enhancing Predictive Capabilities:**

Investigate the potential of Deep Reinforcement Learning (DRL) and real-time adaptive models for improved grid stability management. Explore hybrid approaches that combine statistical modeling, AI, and real-time grid control systems for better predictive accuracy.

4. Literature Review

Combination of optimization and machine learning methods has transformed energy management in renewable energy districts and smart grids. A study suggested a machine learning model based on Gaussian Process Regression (GPR) integrated with Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) to optimize energy surplus, revenue, and cost. This method was superior to conventional approaches but struggled to manage stochastic and seasonal fluctuations. The study pointed out shortcomings in holistic machine learning-based energy management, particularly in bidirectional energy flow optimization and handling seasonal variations.
Energy management and optimization methods for grid-scale energy storage systems (ESS) have been widely researched, especially in maximizing storage efficiency and grid stability. Optimization methods such as Linear Programming (LP) and Mixed-Integer Linear Programming (MILP) have been used to enhance battery energy storage systems (BESS). Although these methods offer a solid framework, they are limited in real-time applications and multi-storage coordination, creating large research gaps in optimizing energy arbitrage and maximizing grid benefits.
Big data analytics has been instrumental in dynamic energy management in smart grids. Studies have investigated the application of high-performance computing and predictive modeling for real-time monitoring of power, load forecasting, and effective demand response. Utilizing smart meter data and SCADA systems, these methods provide scalability and strong predictive analytics. Challenges e.g., real-time processing of data, data protection, and optimization problems are still active research areas. Another study also investigated the combination of big data analytics and machine learning for smart grid management, with emphasis on IoT-based sensor networks and cybersecurity. With

advances in predictive analytics and stability, there are still limitations in real-time processing, security strength, and scalability.

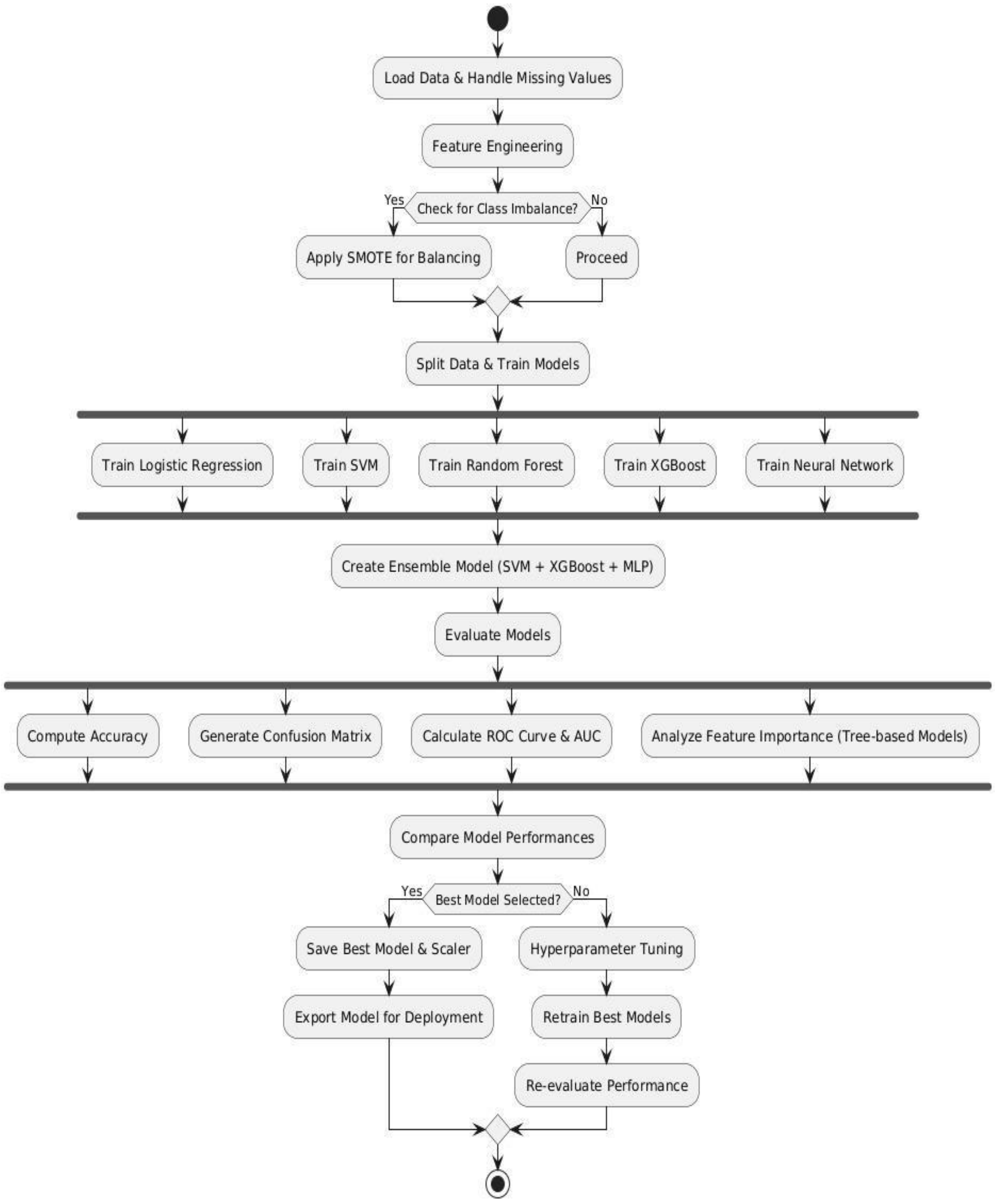


Title	Methodology Used	Strengths	Weaknesses
1.Machine Learning-Based Energy Management Model for Smart Grid and Renewable Energy Districts (2020)	Gaussian Process Regression (GPR), Genetic Algorithms (GA), Particle Swarm Optimization (PSO)	Optimized energy surplus, cost, and revenue	Lack of interpretability, struggles with stochastic and seasonal variations
2.Energy Management and Optimization Methods for Grid Energy Storage Systems (2017)	Review of EMS architectures, BESS applications, and optimization methods (LP, MILP)	Comprehensive review of optimization techniques for storage efficiency and grid stability	Limited real-time applicability, struggles with multi-storage coordination
3.Big Data Analytics for Dynamic Energy Management in Smart Grids (2015)	Big data analytics, predictive modeling, and high-performance computing applied to smart meters and SCADA systems	Real-time data processing, scalability for large datasets, robust data security	Not explicitly mentioned
4.Application of Big Data and Machine Learning in Smart Grid, and Associated Security Concerns: A Review (2018)	Review of smart meter data, IoT-based sensor data, and SCADA system data	IoT integration, cybersecurity measures, predictive analytics for stability	Limited real-time processing, security concerns, scalability issues
5.Decentralized Smart Grid Stability Modeling with Machine Learning (2023)	Machine learning models (MLP, XGB, SVM, GP) applied to a public dataset with 30,000 data points	Effective use of AI, XGB model shows best performance	Lacks real-time control, response efficiency, AI optimization
6.Deep Neural Network-Based Smart Grid Stability Analysis: Enhancing Grid Resilience and Performance (2024)	Machine learning classifiers (Logistic Regression, XGBoost, Linear SVM, SVM-RBF), Deep Neural Networks (DNN)	High accuracy in stability prediction	Lacks real-time adaptability, underutilizes deep learning, lacks scalability

Machine learning has also been employed in decentralized smart grid stability modeling. Various machine learning models like Multilayer Perceptron (MLP), XGBoost (XGB), Support Vector Machine (SVM), and Gaussian Process (GP) have been tested by researchers, where the top performance was delivered by XGB. While AI-based models increase stability, they are not adjusted in real-time and do not simulate interaction, which indicates the future direction in decentralized grid stability.
Deep learning techniques have been explored for enhancing the smartness and resilience of smart grids. Classification techniques such as Logistic Regression, XGBoost, Linear SVM, and SVM-RBF, and Deep Neural Networks (DNN) have shown potential in forecasting grid stability. However, these models still face limitations in real-time adaptability and scalability in large-scale smart grids. There is a growing need for more robust deep models of learning that can accommodate real-time processing and high stability prediction accuracy.

Literature highlights machine learning, big data analytics, and optimization development in smart grid energy management. While significant breakthroughs have already occurred, most challenges remain matters of real-time flexibility, security, scalability, and bidirectional flow of energy optimization. The focus of future work should involve further research in adaptive, scalable, and optimal AI-based models for making the smart grid stable and operationally more efficient.

5. Proposed Methodology



Load Data & Handle Missing Values
Feature Engineering
Check for Class Imbalance?
Yes
No
Apply SMOTE for Balancing
Proceed
Split Data & Train Models
Train Logistic Regression
Train SVM
Train Random Forest
Train XGBoost
Train Neural Network
Create Ensemble Model (SVM + XGBoost + MLP)
Evaluate Models
Compute Accuracy
Generate Confusion Matrix
Calculate ROC Curve & AUC
Analyze Feature Importance (Tree-based Models)
Compare Model Performances
Best Model Selected?
Yes
No
Save Best Model & Scaler
Export Model for Deployment
Hyperparameter Tuning
Retrain Best Models
Re-evaluate Performance

Stepwise Algorithm for Machine Learning Pipeline

Step 1: Load Dataset & Preprocessing

Step 2: Train-Test Split & Class Imbalance Handling

1. Split the dataset into 80% training and 20% testing
2. Apply SMOTE-ENN to the training dataset to handle class imbalance:

Step 3: Model Training & Evaluation

3.1 Support Vector Machine (SVM)

3.2 Random Forest Classifier

3.3 Multilayer Perceptron (MLP)

3.4 Deep Learning Model (Keras + TensorFlow)

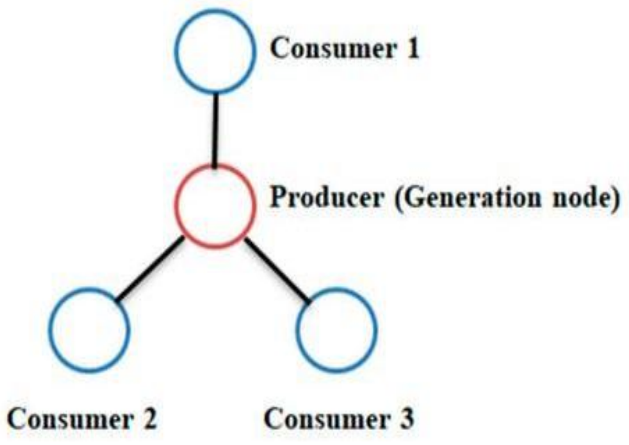
Step 4: Compare Models & Select Best Model

Step 5: Save & Deploy Best Model

Detailed Description of the Dataset:

The data set employed in the research, obtained from the Machine Learning Repository of the University of California, is intended for stability analysis of smart grids. It has 60,000 rows and 14 features, simulating the interactions between a single producer node and three consumer nodes in a four-node star network.

four-node star network.



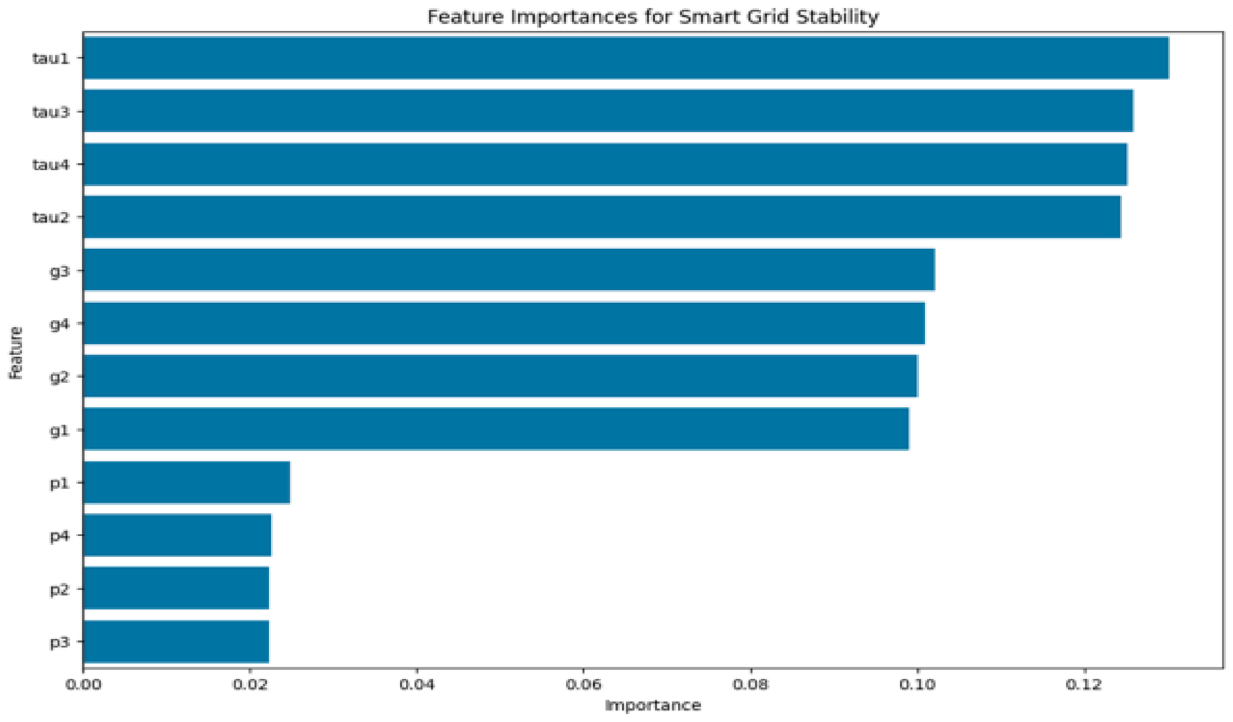
1. **Dataset Overview**

- Total Records: 60,000 instances
- Total Features: 14 (12 independent variables + 2 dependent variables)
- Feature Categories:
- Reaction Time (τ) [4 features] – Measures how fast a producer or consumer responds to changes in energy supply/demand.

- Total Power Balance (P) [4 features] – Indicates whether energy is being produced or consumed.
- Price Elasticity Coefficient (γ) [4 features] – Represents energy price elasticity for each grid participant.
- Stability Labels [2 dependent features] – Defines whether the grid is stable or unstable.

2. **Structure of the dataset:**

- The dataset represents a four-node star network, which includes:
- 1 Producer Node (P1) – Generates electricity and interacts with consumers.
- 3 Consumer Nodes (P2, P3, P4) – Consume electricity based on demand and price elasticity.
- Stability is determined based on the interactions between production, consumption, and reaction times.

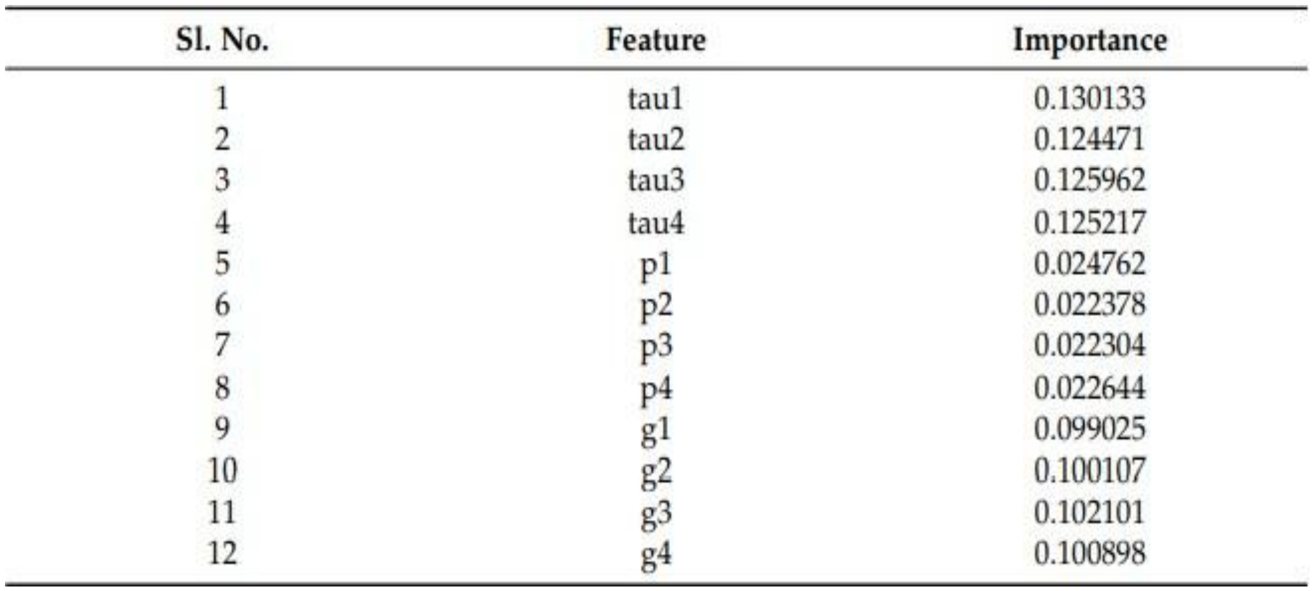


- Key Insights:
- Reaction time (τ) is most influential in determining stability, as slower response times can lead to instability.
- Power balance (P) impacts whether a node is supplying or consuming energy.
- Price elasticity (γ) affects market-driven adjustments in electricity distribution.

3. Stability Labeling Criteria:

- The dataset provides two stability indicators:
- stab (numeric stability value)
- Positive stab value → Unstable grid
- Negative stab value → Stable gridstabf (binary stability label)

- "stable" (1) → Grid is operating normally.
- "unstable" (0) → Grid faces disruptions or power fluctuations.
- The labeling is done based on the differential equation-based decentral smart grid control model, which evaluates the system's behavior under different conditions.



Sl. No.	Feature	Importance
1	tau1	0.130133
2	tau2	0.124471
3	tau3	0.125962
4	tau4	0.125217
5	p1	0.024762
6	p2	0.022378
7	p3	0.022304
8	p4	0.022644
9	g1	0.099025
10	g2	0.100107
11	g3	0.102101
12	g4	0.100898

Preprocessing Steps for Grid Stability Prediction:

1. Identify Feature & Target Columns – Separate independent variables (features) and the dependent variable (target).
2. Categorize Features – Classify features as numerical (e.g., voltage, power) or categorical (e.g., energy source type).
3. Encode Categorical Features – Convert categorical data to numeric using String Indexing and One-Hot Encoding.
4. Encode Target Variable – If the target is categorical, convert it into numerical labels for classification.
5. Feature Vectorization – Combine all features into a single numerical vector for model input.
6. Standardization – Scale features to have zero mean and unit variance for better model performance.
7. Train-Test Split – Divide the data into 80% training and 20% testing for model evaluation.
8. Handle Class Imbalance (SMOTE-ENN) – Balance dataset by oversampling minority class and removing noisy data.
9. Prepare Final Data – Convert processed data into PySpark format for model training.

2. Model Training Pipeline

Here, we describe training and testing of these machine learning models, i.e., Support Vector Machine (SVM), Random Forest Classifier, Multilayer Perceptron (MLP), and Deep Learning Model (Keras + TensorFlow). Each of these models has been trained and optimized to achieve the maximum possible classification accuracy on the dataset.
The dataset contained a large number of training and test instances. SMOTE-ENN was employed in order to ensure that the models ran effectively and generalized well for the sake of ensuring classification accuracy. The algorithm solved class imbalance by generating artificial instances of the minority class but removing noisy and borderline samples, thus improving the classification accuracy. All models were trained via multiple training cycles, hyperparameter adjustment, testing for performance, and optimization in order to meet the best performance. Models' performance was assessed with various assessment measures, like accuracy, precision, recall, F1-score, confusion

(i) Support Vector Machine (SVM)

Support Vector Machine (SVM) is a classification algorithm of general purpose that finds the optimal hyperplane to distinguish classes in a high-dimensional feature space. The model was trained using 3-fold cross-validation, which split the data into three subsets. One subset was used as a validation set and the remaining were used for training a single time. This enhanced generalization and prevented overfitting.
To test different decision boundaries, different kernel functions were experimented with. Linear Kernel was initially employed to observe if there was a linear separability between classes or not. Due to the fact that real data is rarely available in linearly separable form, the RBF Kernel and Polynomial Kernel were also experimented with. These non-linear kernels helped in projecting the data in higher dimensional spaces where linear separability could be achieved better.
Hyperparameter tuning was performed to tune the C parameter (used for balancing the trade-off between having low error and having maximum margin) and Gamma (in the case of RBF Kernel), which controls the impact of a single data point on the decision boundary. Grid Search and Randomized Search approaches were employed to find optimal hyperparameter values.
After training, model performance was evaluated with precision, recall, and F1-score, all of which are particularly relevant for handling class imbalance. A confusion matrix was used to examine the number of correctly and incorrectly labeled instances, giving a better insight into model performance.

(ii) Random Forest Classifier

Random Forest Classifier is an ensemble learner that creates many decision trees and combines their outputs to reduce variance and increase accuracy. It adds robustness and prevents overfitting, thus being a highly effective classifying method.
The model was trained with 3-fold cross-validation, just like SVM, to allow for better generalization. Hyperparameters were also optimized, such as the number of trees (n_estimators) and maximum depth (max_depth) of a single tree. The depth of every tree was particularly optimized to maintain learning complexity without overfitting.
One of the better features of the Random Forest model is that it can return feature importance scores. By considering what features were most responsible for the classification, it could then tweak the model by using the most important features and discarding less important ones. This improved interpretability and simplified the model.
The model performance was tested using a confusion matrix, which indicated the true positives, true negatives, false positives, and false negatives. The ROC-AUC curve was also tested to determine the ability of the model to differentiate between classes. A classification report with precision, recall, and F1-score was also produced to determine how effectively the model performed on both classes.

(iii) **Multilayer Perceptron (MLP)**

The Multilayer Perceptron (MLP) is an artificial neural network (ANN) model that is used to learn intricate patterns in data. The model was designed with several hidden layers, and each of them employed the ReLU activation function to bring in non-linearity and avoid vanishing gradient problems. The output layer employed the Softmax activation function for multi-class classification. Training was conducted with backpropagation and gradient descent optimization, such that weights were iteratively updated to reduce classification errors. To improve the performance and stability of the model, methods like batch normalization and dropout layers were used. Batch normalization normalized the activations in layers, which improved convergence, while dropout layers avoided overfitting by randomly disabling neurons during training.

Hyperparameter tuning required adjusting the number of neurons in each layer, the learning rate, batch size, and the number of epochs. The optimization was attempted by experimenting with various values and observing their impact on the model's learning process.

Model assessment involved verifying the confusion matrix for misclassifications, precision, recall, and F1-score, and observing the convergence of the loss function. Overfitting issues were identified and resolved by comparing the training loss and validation loss of the model.

(iv) Deep Learning Model (Keras + TensorFlow)

We used a DNN model in Keras using the TensorFlow backend. The model had three hidden fully connected layers of neurons in the architecture of 128-64-32. The ReLU activation function is been used in the hidden layers to bring in non-linearity, and the Softmax/Sigmoid activation function was used in the output layer to generate probability-based outputs.

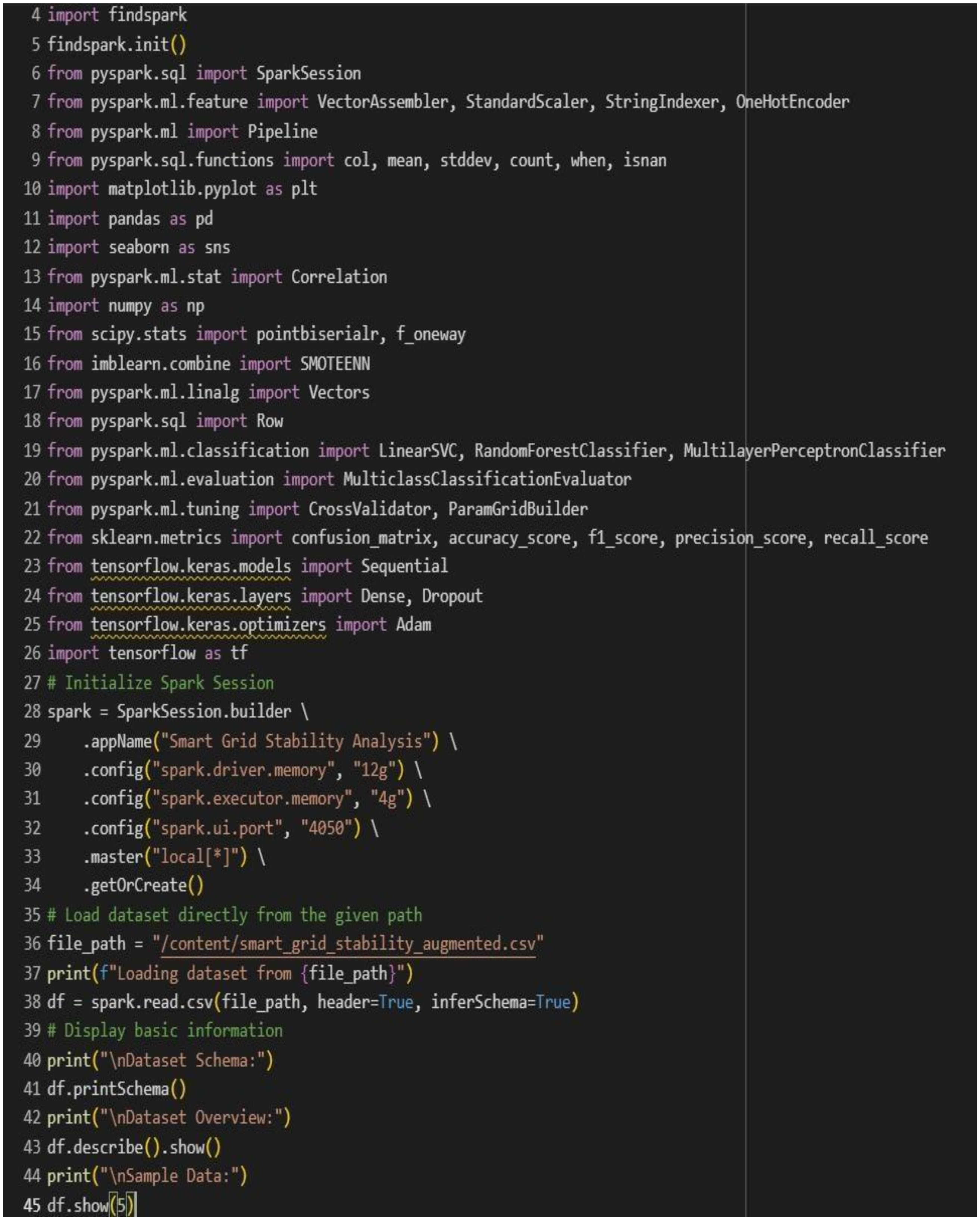
To promote model generalization and avoid overfitting, dropout layers were added following every hidden layer. L2 regularization was also utilized to regularize large weights so that a stable and smooth training process was guaranteed. The Adam optimizer was used to optimize the model, where learning rate was adaptively scaled to achieve maximum speed of convergence.

Hyperparameter tuning of the learning rate, number of epochs, and batch size was incorporated in training. Different values were tried to observe how they impacted convergence during training as well as model performance. Learning curve plots of the model were performed such that the training loss reduced but the validation loss remained constant.

For testing, cross-entropy loss was employed as the loss metric and precision-recall curves were employed as the metrics. Accuracy between training and validation was graphed to evaluate how well the model performs on new data. Confusion matrix was also employed to measure correct and incorrect predictions.

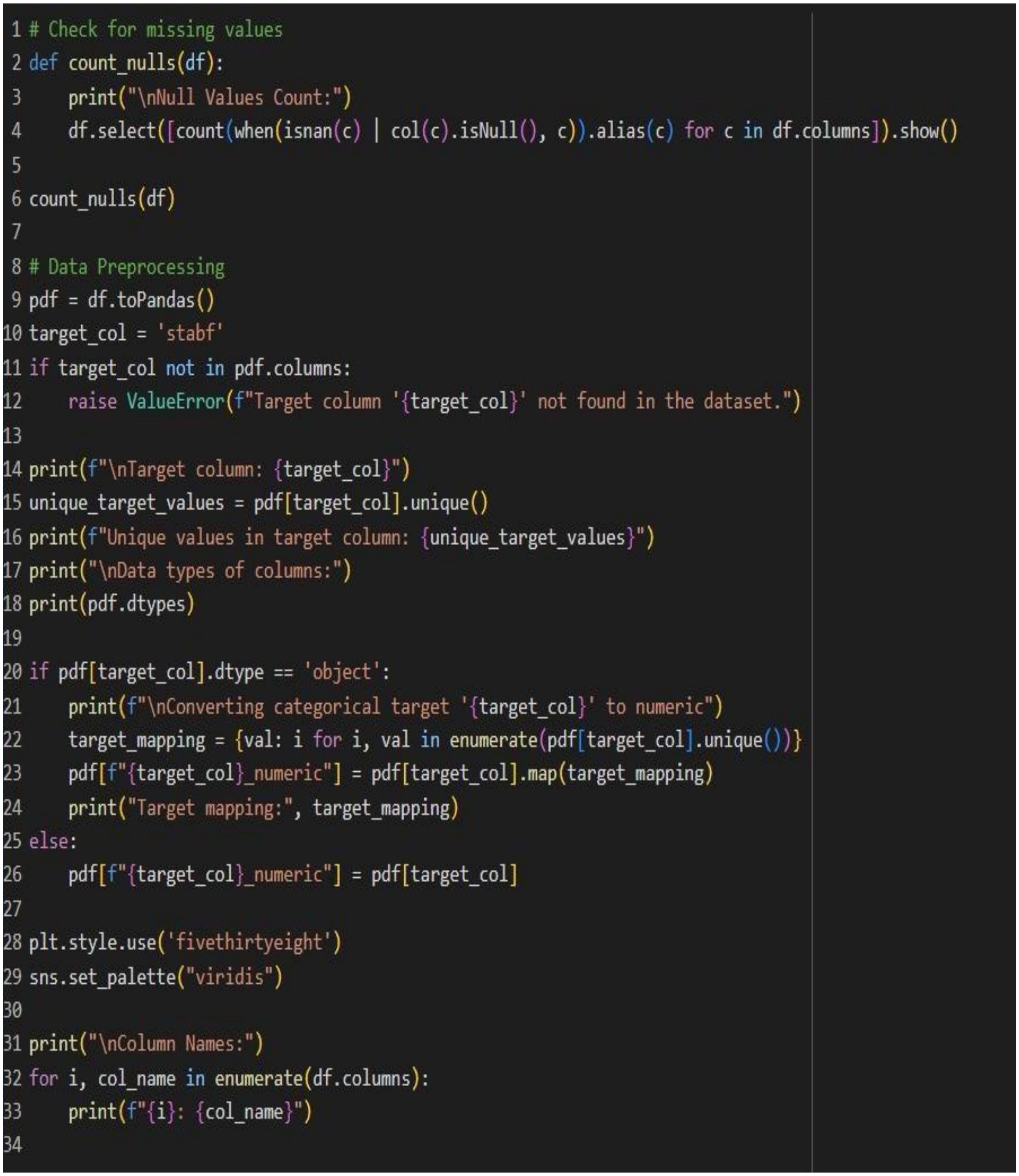
6. Implementation:

6.1. Data loading:



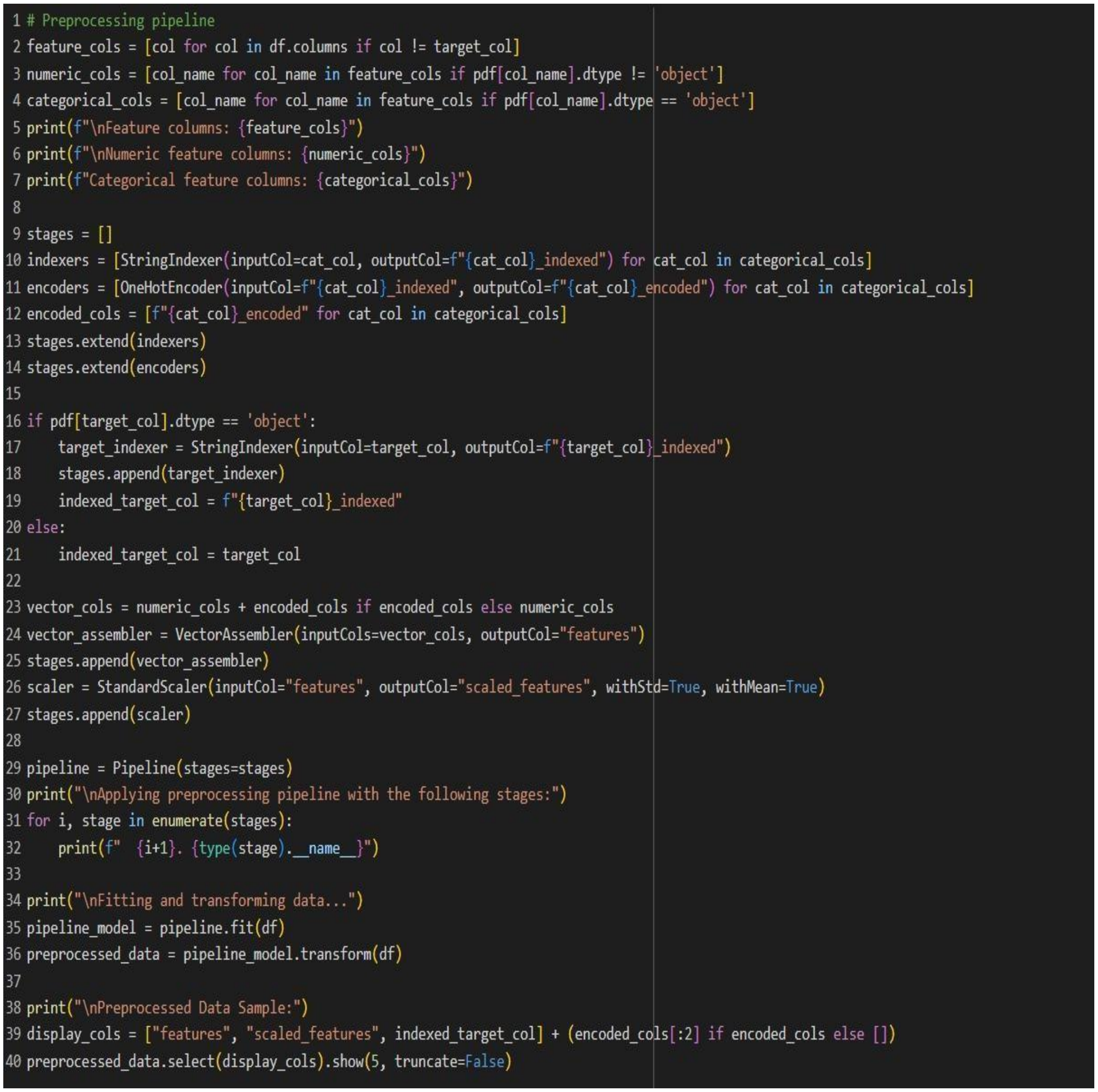
```python
import findspark
findspark.init()
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StandardScaler, StringIndexer, OneHotEncoder
from pyspark.ml import Pipeline
from pyspark.sql.functions import col, mean, stddev, count, when, isnan
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from pyspark.ml.stat import Correlation
import numpy as np
from scipy.stats import pointbiserialr, f_oneway
from imblearn.combine import SMOTEENN
from pyspark.ml.linalg import Vectors
from pyspark.sql import Row
from pyspark.ml.classification import LinearSVC, RandomForestClassifier, MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, recall_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
# Initialize Spark Session
spark = SparkSession.builder \
    .appName("Smart Grid Stability Analysis") \
    .config("spark.driver.memory", "12g") \
    .config("spark.executor.memory", "4g") \
    .config("spark.ui.port", "4050") \
    .master("local[*]") \
    .getOrCreate()
# Load dataset directly from the given path
file_path = "/content/smart_grid_stability_augmented.csv"
print(f"Loading dataset from {file_path}")
df = spark.read.csv(file_path, header=True, inferSchema=True)
# Display basic information
print("\nDataset Schema:")
df.printSchema()
print("\nDataset Overview:")
df.describe().show()
print("\nSample Data:")
df.show(5)
```

6.2. Data processing:



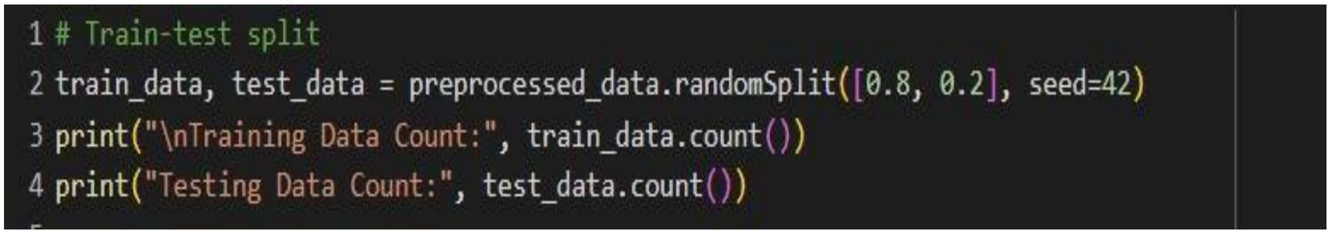
```python
# Check for missing values
def count_nulls(df):
    print("\nNull Values Count:")
    df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df.columns]).show()

count_nulls(df)

# Data Preprocessing
pdf = df.toPandas()
target_col = 'stabf'
if target_col not in pdf.columns:
    raise ValueError(f"Target column '{target_col}' not found in the dataset.")

print(f"\nTarget column: {target_col}")
unique_target_values = pdf[target_col].unique()
print(f"Unique values in target column: {unique_target_values}")
print("\nData types of columns:")
print(pdf.dtypes)

if pdf[target_col].dtype == 'object':
    print(f"\nConverting categorical target '{target_col}' to numeric")
    target_mapping = {val: i for i, val in enumerate(pdf[target_col].unique())}
    pdf[f"{target_col}_numeric"] = pdf[target_col].map(target_mapping)
    print("Target mapping:", target_mapping)
else:
    pdf[f"{target_col}_numeric"] = pdf[target_col]

plt.style.use('fivethirtyeight')
sns.set_palette("viridis")

print("\nColumn Names:")
for i, col_name in enumerate(df.columns):
    print(f"{i}: {col_name}")

```

6.3. Data preprocessing:



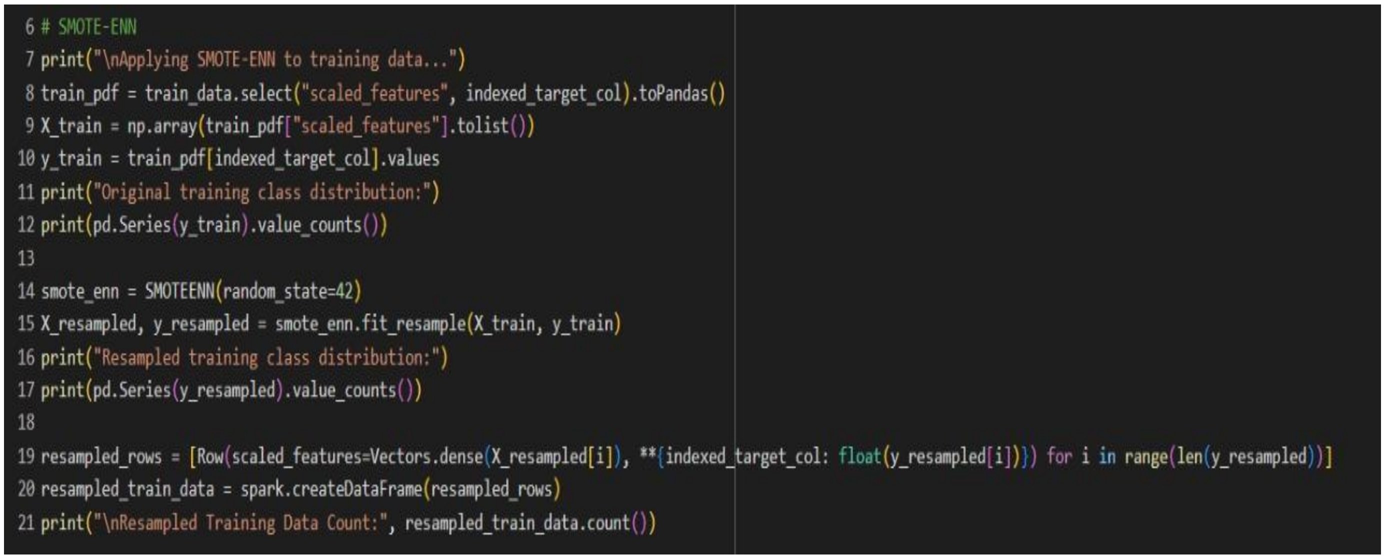
```python
# Preprocessing pipeline
feature_cols = [col for col in df.columns if col != target_col]
numeric_cols = [col_name for col_name in feature_cols if pdf[col_name].dtype != 'object']
categorical_cols = [col_name for col_name in feature_cols if pdf[col_name].dtype == 'object']
print(f"\nFeature columns: {feature_cols}")
print(f"\nNumeric feature columns: {numeric_cols}")
print(f"Categorical feature columns: {categorical_cols}")

stages = []
indexers = [StringIndexer(inputCol=cat_col, outputCol=f"{cat_col}_indexed") for cat_col in categorical_cols]
encoders = [OneHotEncoder(inputCol=f"{cat_col}_indexed", outputCol=f"{cat_col}_encoded") for cat_col in categorical_cols]
encoded_cols = [f"{cat_col}_encoded" for cat_col in categorical_cols]
stages.extend(indexers)
stages.extend(encoders)

if pdf[target_col].dtype == 'object':
    target_indexer = StringIndexer(inputCol=target_col, outputCol=f"{target_col}_indexed")
    stages.append(target_indexer)
    indexed_target_col = f"{target_col}_indexed"
else:
    indexed_target_col = target_col

vector_cols = numeric_cols + encoded_cols if encoded_cols else numeric_cols
vector_assembler = VectorAssembler(inputCols=vector_cols, outputCol="features")
stages.append(vector_assembler)
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=True, withMean=True)
stages.append(scaler)

pipeline = Pipeline(stages=stages)
print("\nApplying preprocessing pipeline with the following stages:")
for i, stage in enumerate(stages):
    print(f"  {i+1}. {type(stage).__name__}")

print("\nFitting and transforming data...")
pipeline_model = pipeline.fit(df)
preprocessed_data = pipeline_model.transform(df)

print("\nPreprocessed Data Sample:")
display_cols = ["features", "scaled_features", indexed_target_col] + (encoded_cols[:2] if encoded_cols else [])
preprocessed_data.select(display_cols).show(5, truncate=False)
```

6.4. splitting data:



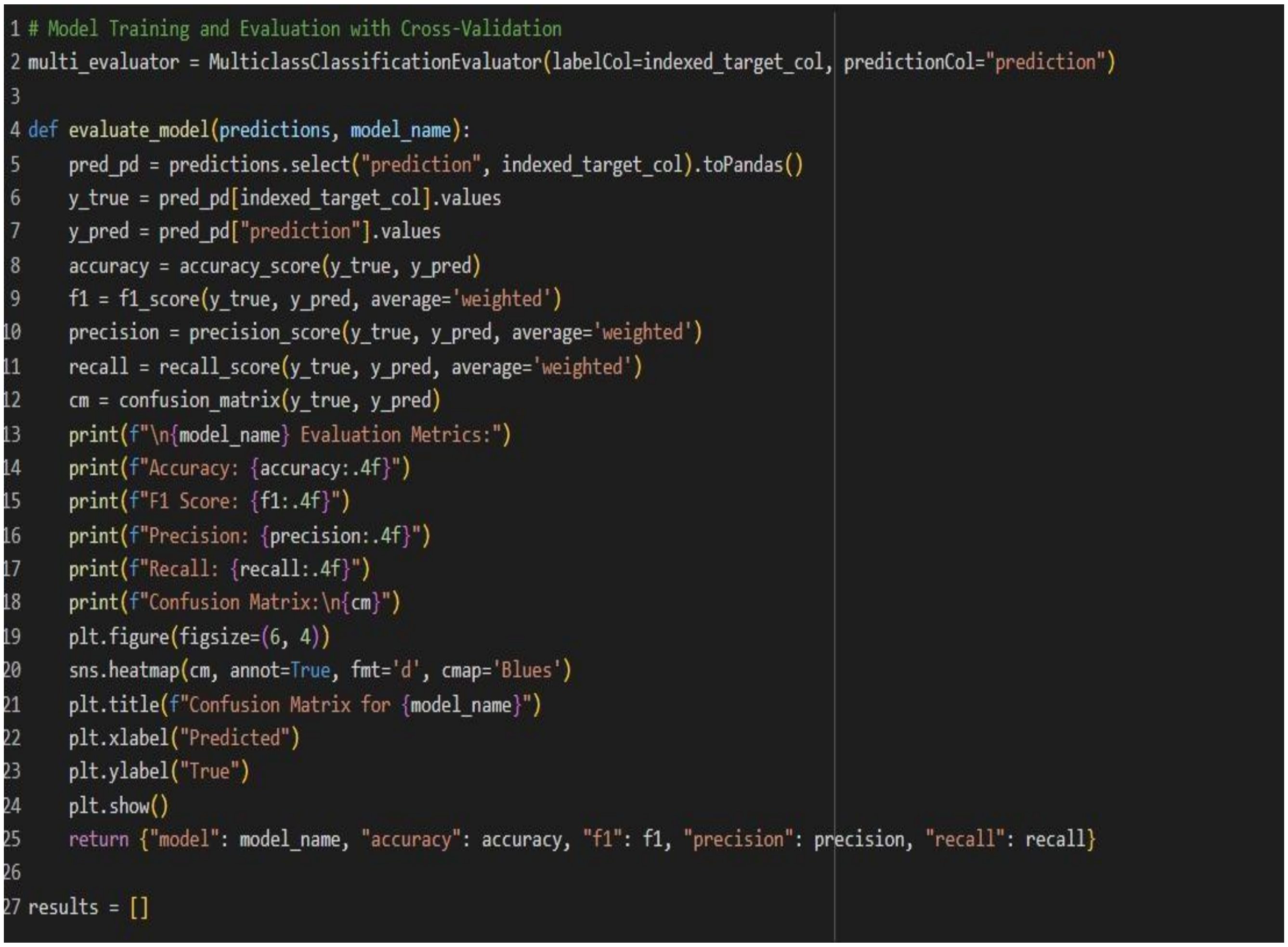
```python
# Train-test split
train_data, test_data = preprocessed_data.randomSplit([0.8, 0.2], seed=42)
print("\nTraining Data Count:", train_data.count())
print("Testing Data Count:", test_data.count())
```

6.5. Balancing data:



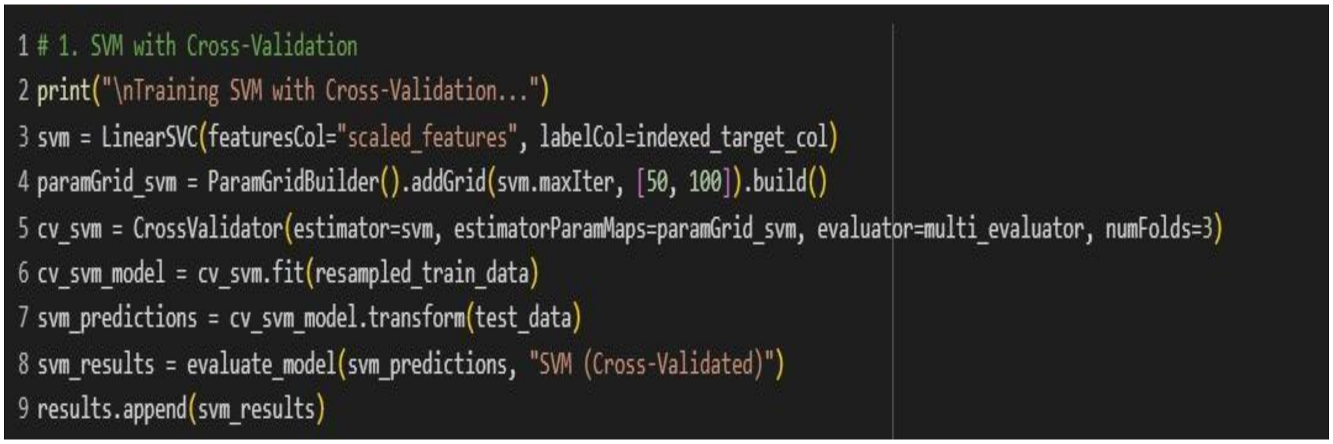
```python
# SMOTE-ENN
print("\nApplying SMOTE-ENN to training data...")
train_pdf = train_data.select("scaled_features", indexed_target_col).toPandas()
X_train = np.array(train_pdf["scaled_features"].tolist())
y_train = train_pdf[indexed_target_col].values
print("Original training class distribution:")
print(pd.Series(y_train).value_counts())

smote_enn = SMOTEENN(random_state=42)
X_resampled, y_resampled = smote_enn.fit_resample(X_train, y_train)
print("Resampled training class distribution:")
print(pd.Series(y_resampled).value_counts())

resampled_rows = [Row(scaled_features=Vectors.dense(X_resampled[i]), **{indexed_target_col: float(y_resampled[i])}) for i in range(len(y_resampled))]
resampled_train_data = spark.createDataFrame(resampled_rows)
print("\nResampled Training Data Count:", resampled_train_data.count())
```

6.6. Evaluate model function:



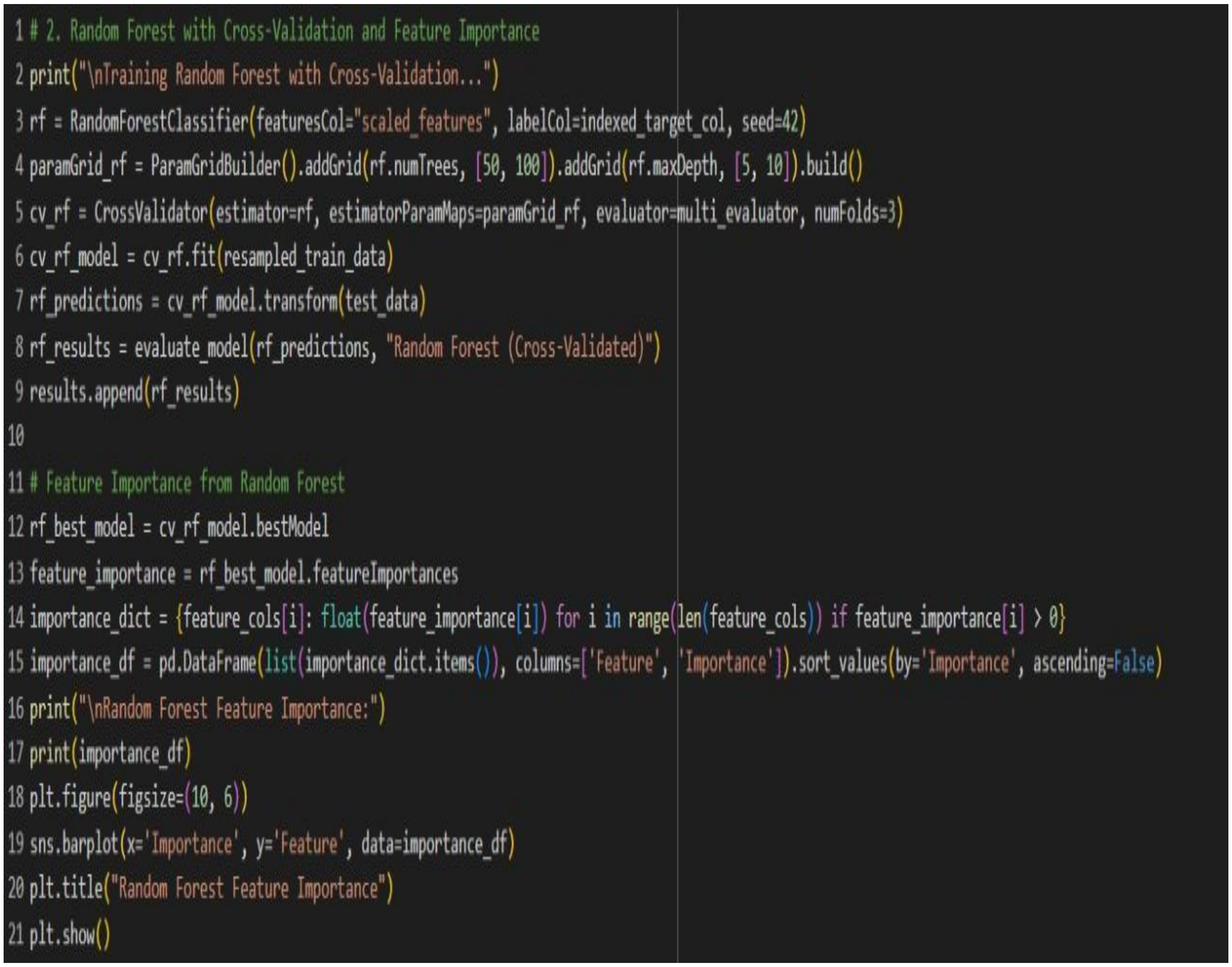
```python
# Model Training and Evaluation with Cross-Validation
multi_evaluator = MulticlassClassificationEvaluator(labelCol=indexed_target_col, predictionCol="prediction")

def evaluate_model(predictions, model_name):
    pred_pd = predictions.select("prediction", indexed_target_col).toPandas()
    y_true = pred_pd[indexed_target_col].values
    y_pred = pred_pd["prediction"].values
    accuracy = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred, average='weighted')
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    cm = confusion_matrix(y_true, y_pred)
    print(f"\n{model_name} Evaluation Metrics:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"F1 Score: {f1:.4f}")
    print(f"Precision: {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    print(f"Confusion Matrix:\n{cm}")
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"Confusion Matrix for {model_name}")
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()
    return {"model": model_name, "accuracy": accuracy, "f1": f1, "precision": precision, "recall": recall}

results = []
```

6.7. SVM:



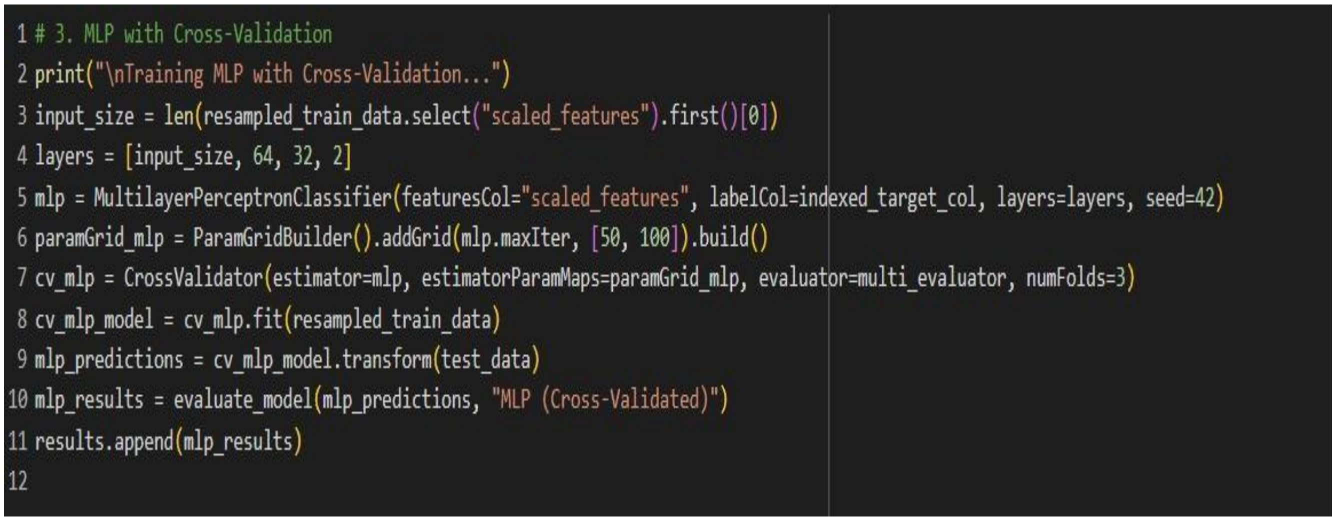
```
# 1. SVM with Cross-Validation
print("\nTraining SVM with Cross-Validation...")
svm = LinearSVC(featuresCol="scaled_features", labelCol=indexed_target_col)
paramGrid_svm = ParamGridBuilder().addGrid(svm.maxIter, [50, 100]).build()
cv_svm = CrossValidator(estimator=svm, estimatorParamMaps=paramGrid_svm, evaluator=multi_evaluator, numFolds=3)
cv_svm_model = cv_svm.fit(resampled_train_data)
svm_predictions = cv_svm_model.transform(test_data)
svm_results = evaluate_model(svm_predictions, "SVM (Cross-Validated)")
results.append(svm_results)
```

6.8. Random Forest:



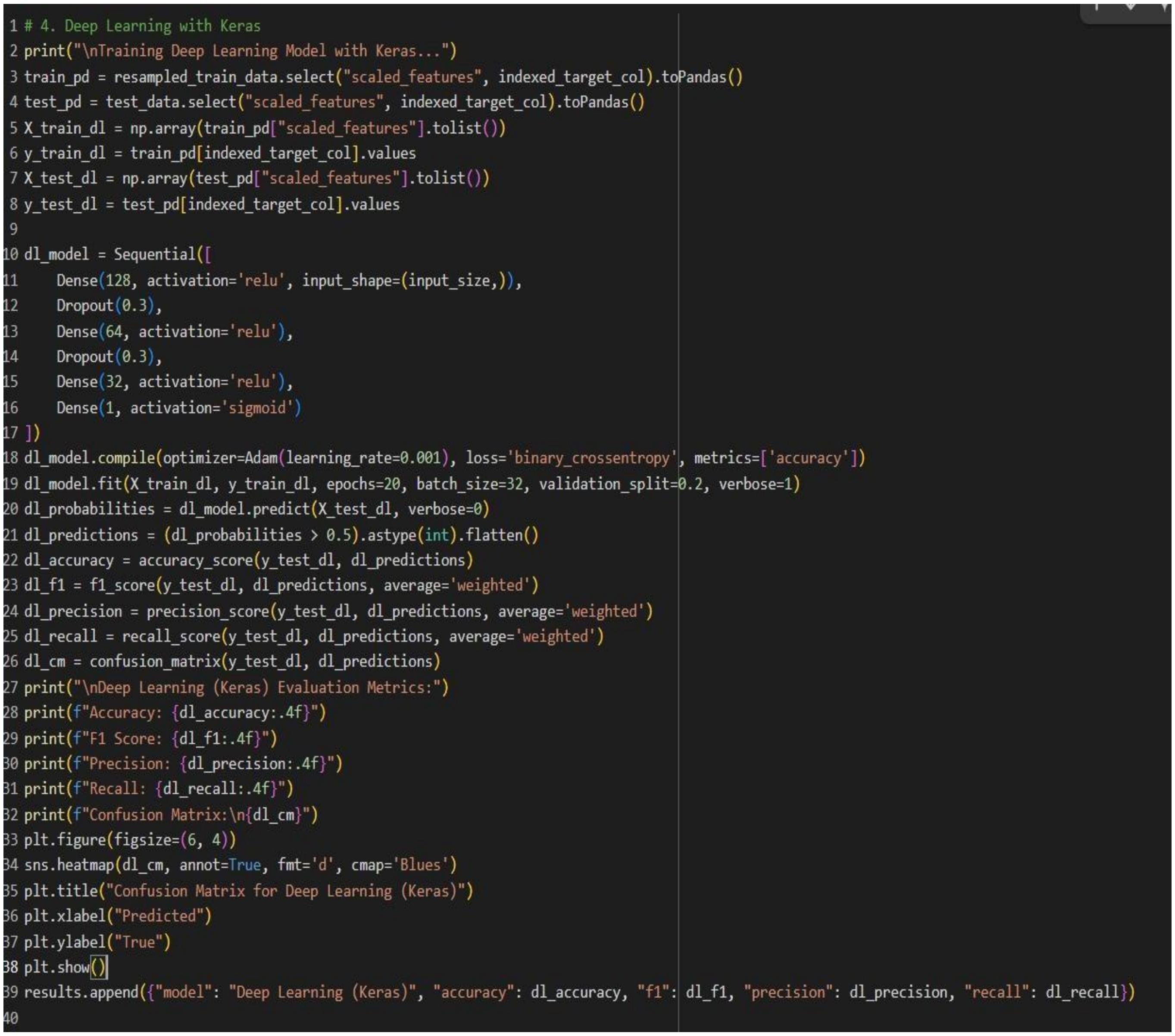
```
# 2. Random Forest with Cross-Validation and Feature Importance
print("\nTraining Random Forest with Cross-Validation...")
rf = RandomForestClassifier(featuresCol="scaled_features", labelCol=indexed_target_col, seed=42)
paramGrid_rf = ParamGridBuilder().addGrid(rf.numTrees, [50, 100]).addGrid(rf.maxDepth, [5, 10]).build()
cv_rf = CrossValidator(estimator=rf, estimatorParamMaps=paramGrid_rf, evaluator=multi_evaluator, numFolds=3)
cv_rf_model = cv_rf.fit(resampled_train_data)
rf_predictions = cv_rf_model.transform(test_data)
rf_results = evaluate_model(rf_predictions, "Random Forest (Cross-Validated)")
results.append(rf_results)

# Feature Importance from Random Forest
rf_best_model = cv_rf_model.bestModel
feature_importance = rf_best_model.featureImportances
importance_dict = {feature_cols[i]: float(feature_importance[i]) for i in range(len(feature_cols)) if feature_importance[i] > 0}
importance_df = pd.DataFrame(list(importance_dict.items()), columns=['Feature', 'Importance']).sort_values(by='Importance', ascending=False)
print("\nRandom Forest Feature Importance:")
print(importance_df)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title("Random Forest Feature Importance")
plt.show()
```

6.9. MultilayerPerceptron:



```
# 3. MLP with Cross-Validation
print("\nTraining MLP with Cross-Validation...")
input_size = len(resampled_train_data.select("scaled_features").first()[0])
layers = [input_size, 64, 32, 2]
mlp = MultilayerPerceptronClassifier(featuresCol="scaled_features", labelCol=indexed_target_col, layers=layers, seed=42)
paramGrid_mlp = ParamGridBuilder().addGrid(mlp.maxIter, [50, 100]).build()
cv_mlp = CrossValidator(estimator=mlp, estimatorParamMaps=paramGrid_mlp, evaluator=multi_evaluator, numFolds=3)
cv_mlp_model = cv_mlp.fit(resampled_train_data)
mlp_predictions = cv_mlp_model.transform(test_data)
mlp_results = evaluate_model(mlp_predictions, "MLP (Cross-Validated)")
results.append(mlp_results)

```

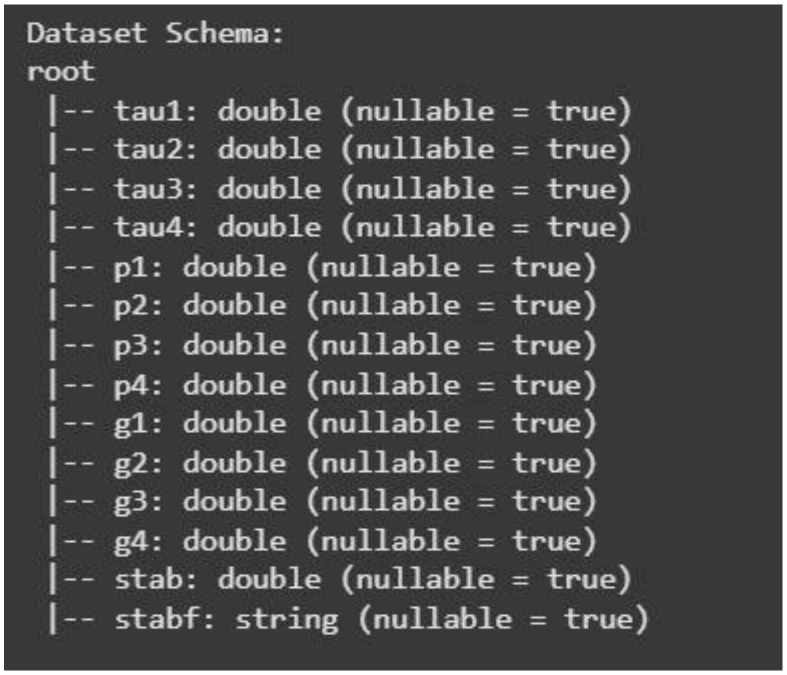
6.10. DeepLearning with Kears:



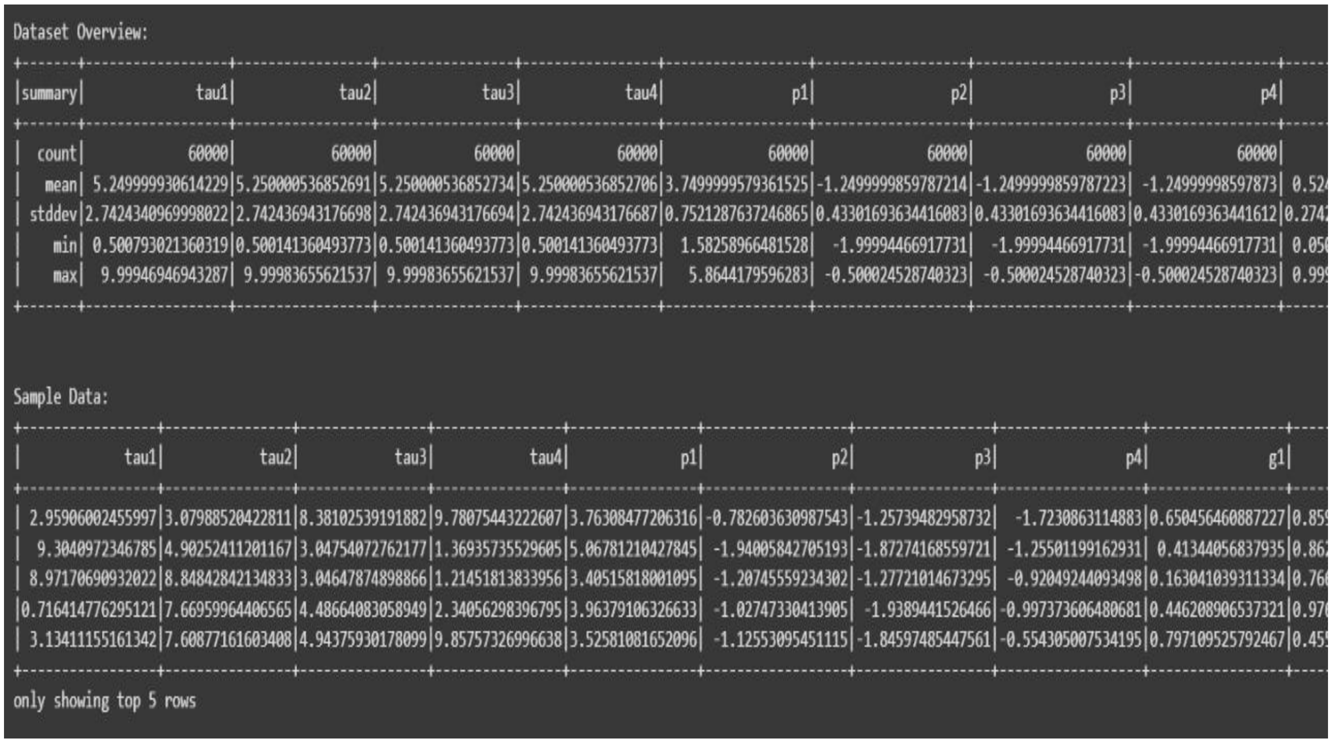
```
# 4. Deep Learning with Keras
print("\nTraining Deep Learning Model with Keras...")
train_pd = resampled_train_data.select("scaled_features", indexed_target_col).toPandas()
test_pd = test_data.select("scaled_features", indexed_target_col).toPandas()
X_train_dl = np.array(train_pd["scaled_features"].tolist())
y_train_dl = train_pd[indexed_target_col].values
X_test_dl = np.array(test_pd["scaled_features"].tolist())
y_test_dl = test_pd[indexed_target_col].values

dl_model = Sequential([
    Dense(128, activation='relu', input_shape=(input_size,)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
dl_model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
dl_model.fit(X_train_dl, y_train_dl, epochs=20, batch_size=32, validation_split=0.2, verbose=1)
dl_probabilities = dl_model.predict(X_test_dl, verbose=0)
dl_predictions = (dl_probabilities > 0.5).astype(int).flatten()
dl_accuracy = accuracy_score(y_test_dl, dl_predictions)
dl_f1 = f1_score(y_test_dl, dl_predictions, average='weighted')
dl_precision = precision_score(y_test_dl, dl_predictions, average='weighted')
dl_recall = recall_score(y_test_dl, dl_predictions, average='weighted')
dl_cm = confusion_matrix(y_test_dl, dl_predictions)
print("\nDeep Learning (Keras) Evaluation Metrics:")
print(f"Accuracy: {dl_accuracy:.4f}")
print(f"F1 Score: {dl_f1:.4f}")
print(f"Precision: {dl_precision:.4f}")
print(f"Recall: {dl_recall:.4f}")
print(f"Confusion Matrix:\n{dl_cm}")
plt.figure(figsize=(6, 4))
sns.heatmap(dl_cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix for Deep Learning (Keras)")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
results.append({"model": "Deep Learning (Keras)", "accuracy": dl_accuracy, "f1": dl_f1, "precision": dl_precision, "recall": dl_recall})

```

7. Results and Discussion:

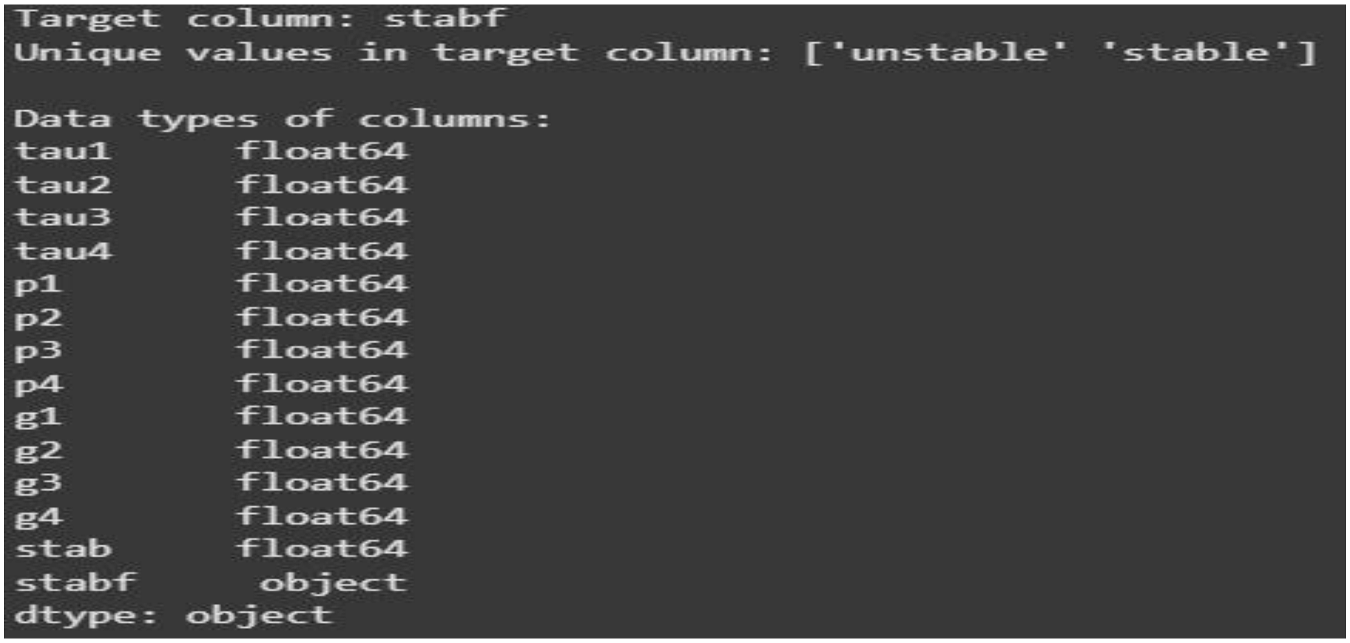
7.1. Dataset Description:



```
Dataset Schema:
root
 |-- tau1: double (nullable = true)
 |-- tau2: double (nullable = true)
 |-- tau3: double (nullable = true)
 |-- tau4: double (nullable = true)
 |-- p1: double (nullable = true)
 |-- p2: double (nullable = true)
 |-- p3: double (nullable = true)
 |-- p4: double (nullable = true)
 |-- g1: double (nullable = true)
 |-- g2: double (nullable = true)
 |-- g3: double (nullable = true)
 |-- g4: double (nullable = true)
 |-- stab: double (nullable = true)
 |-- stabf: string (nullable = true)
```

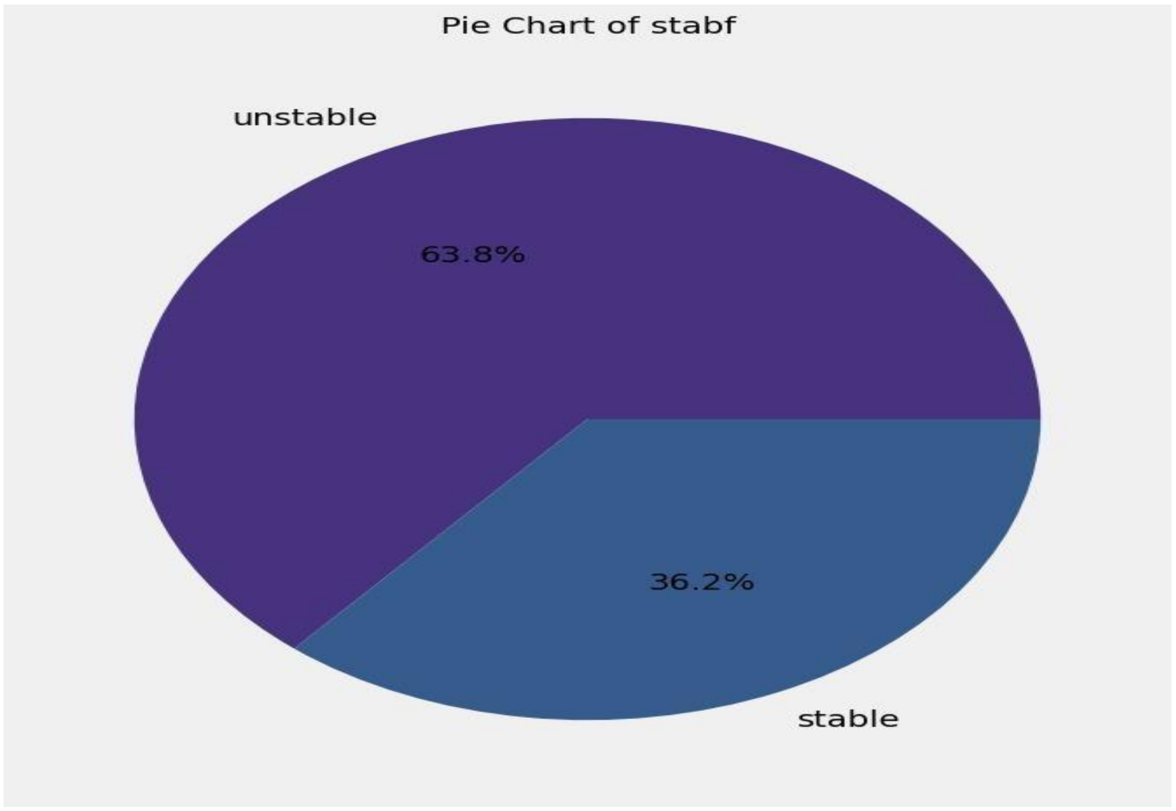


```
Dataset Overview:
+-------+------------------+------------------+------------------+------------------+------------------+-------------------+-------------------+-------------------+-----
|summary|              tau1|              tau2|              tau3|              tau4|                p1|                 p2|                 p3|                 p4|
+-------+------------------+------------------+------------------+------------------+------------------+-------------------+-------------------+-------------------+-----
|  count|             60000|             60000|             60000|             60000|             60000|              60000|              60000|              60000|
|   mean| 5.249999930614229| 5.250000536852691| 5.250000536852734| 5.250000536852706|3.7499999579361525|-1.2499999859787214|-1.2499999859787223|  -1.24999998597873| 0.524
| stddev|2.7424340969998022| 2.742436943176698| 2.742436943176694| 2.742436943176687|0.7521287637246865| 0.4330169363441608| 0.4330169363441608| 0.4330169363441612|0.2742
|    min| 0.500793021360319| 0.500141360493773| 0.500141360493773| 0.500141360493773|  1.58258966481528|  -1.99994466917731|  -1.99994466917731|  -1.99994466917731| 0.050
|    max|  9.99946946943287|  9.99983655621537|  9.99983655621537|  9.99983655621537|   5.8644179596283|-0.500024528740323 |-0.500024528740323 |-0.500024528740323 | 0.999
+-------+------------------+------------------+------------------+------------------+------------------+-------------------+-------------------+-------------------+-----


Sample Data:
+-----------------+-----------------+-----------------+-----------------+-----------------+------------------+-----------------+------------------+-----------------+-----
|             tau1|             tau2|             tau3|             tau4|               p1|                p2|               p3|                p4|               g1|
+-----------------+-----------------+-----------------+-----------------+-----------------+------------------+-----------------+------------------+-----------------+-----
| 2.95906002455997| 3.07988520422811| 8.38102539191882| 9.78075443222607| 3.76308477206316|-0.782603630987543|-1.25739482958732|   -1.7230863114883|0.650456460887227|0.859
|  9.3040972346785| 4.90252411201167| 3.04754072762177| 1.36935735529605| 5.06781210427845| -1.94005842705193|-1.87274168559721|  -1.25501199162931| 0.41344056837935|0.862
| 8.97170690932022| 8.84842842134833| 3.04647874898866| 1.21451813833956| 3.40515818001095| -1.20745559234302|-1.27721014673295|  -0.92049244093498|0.163041039311334|0.766
|0.716414776295121| 7.66959964406565| 4.48664083058949| 2.34056298396795| 3.96379106326633| -1.02747330413905| -1.9389441526466|-0.997373606480681|0.446208906537321|0.976
| 3.13411155161342| 7.60877161603408| 4.94375930178099| 9.85757326996638| 3.52581081652096| -1.12553095451115|-1.84597485447561|-0.554305007534195|0.797109525792467|0.455
+-----------------+-----------------+-----------------+-----------------+-----------------+------------------+-----------------+------------------+-----------------+-----
only showing top 5 rows
```

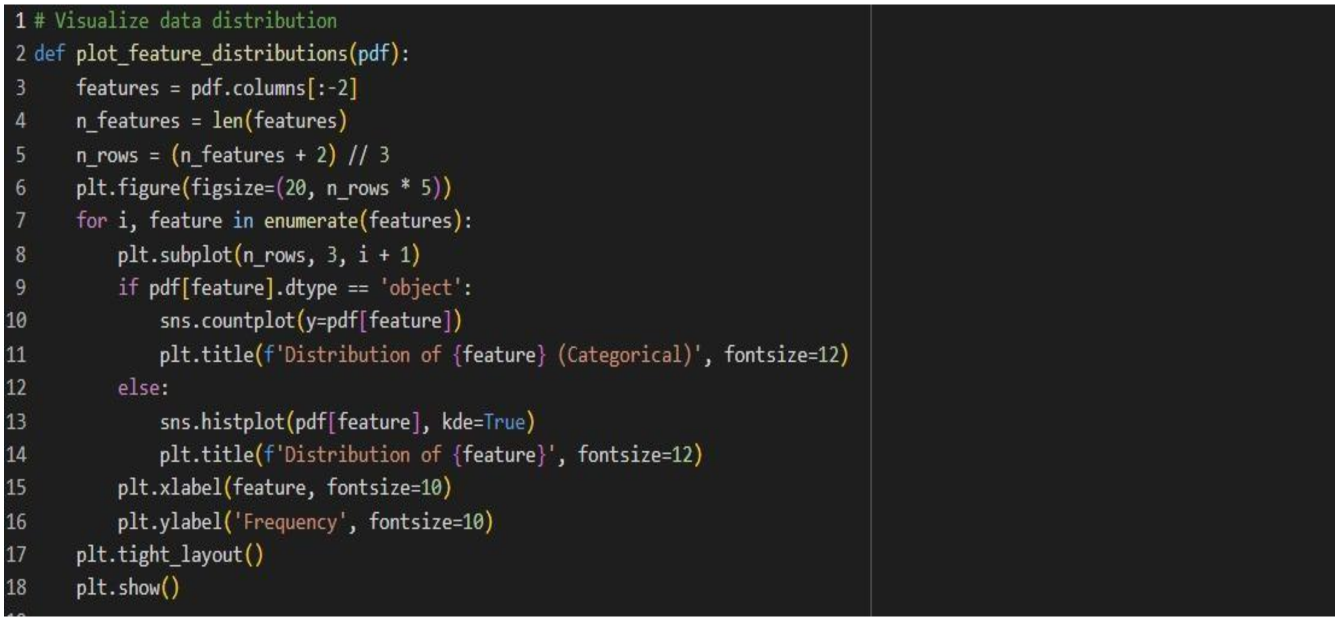


```
Target column: stabf
Unique values in target column: ['unstable' 'stable']

Data types of columns:
tau1      float64
tau2      float64
tau3      float64
tau4      float64
p1        float64
p2        float64
p3        float64
p4        float64
g1        float64
g2        float64
g3        float64
g4        float64
stab      float64
stabf      object
dtype: object
```

TARGET COLUMN DISTRIBUTION:



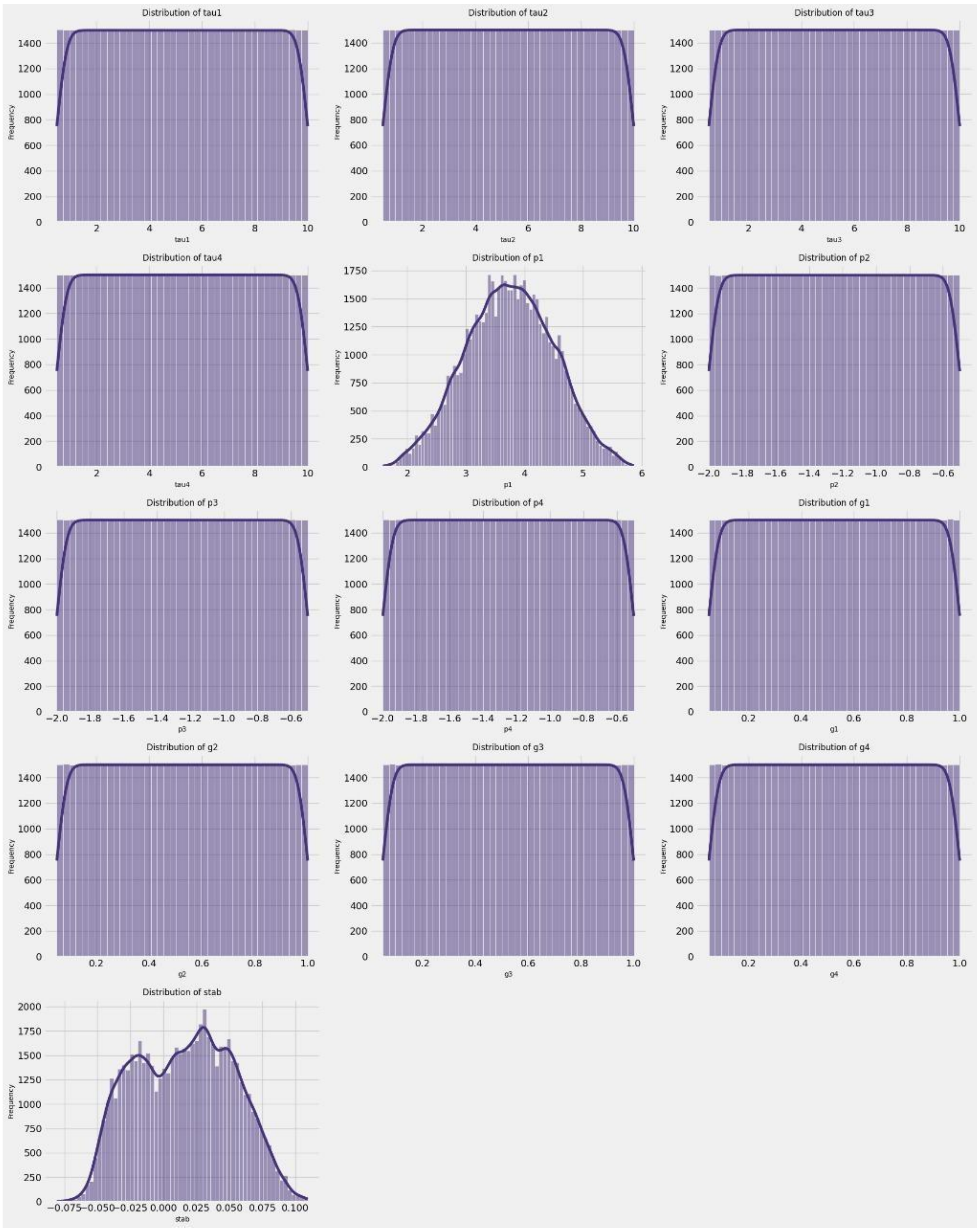
7.2. Feature distribution: CODE:



```python
# Visualize data distribution
def plot_feature_distributions(pdf):
    features = pdf.columns[:-2]
    n_features = len(features)
    n_rows = (n_features + 2) // 3
    plt.figure(figsize=(20, n_rows * 5))
    for i, feature in enumerate(features):
        plt.subplot(n_rows, 3, i + 1)
        if pdf[feature].dtype == 'object':
            sns.countplot(y=pdf[feature])
            plt.title(f'Distribution of {feature} (Categorical)', fontsize=12)
        else:
            sns.histplot(pdf[feature], kde=True)
            plt.title(f'Distribution of {feature}', fontsize=12)
        plt.xlabel(feature, fontsize=10)
        plt.ylabel('Frequency', fontsize=10)
    plt.tight_layout()
    plt.show()
```

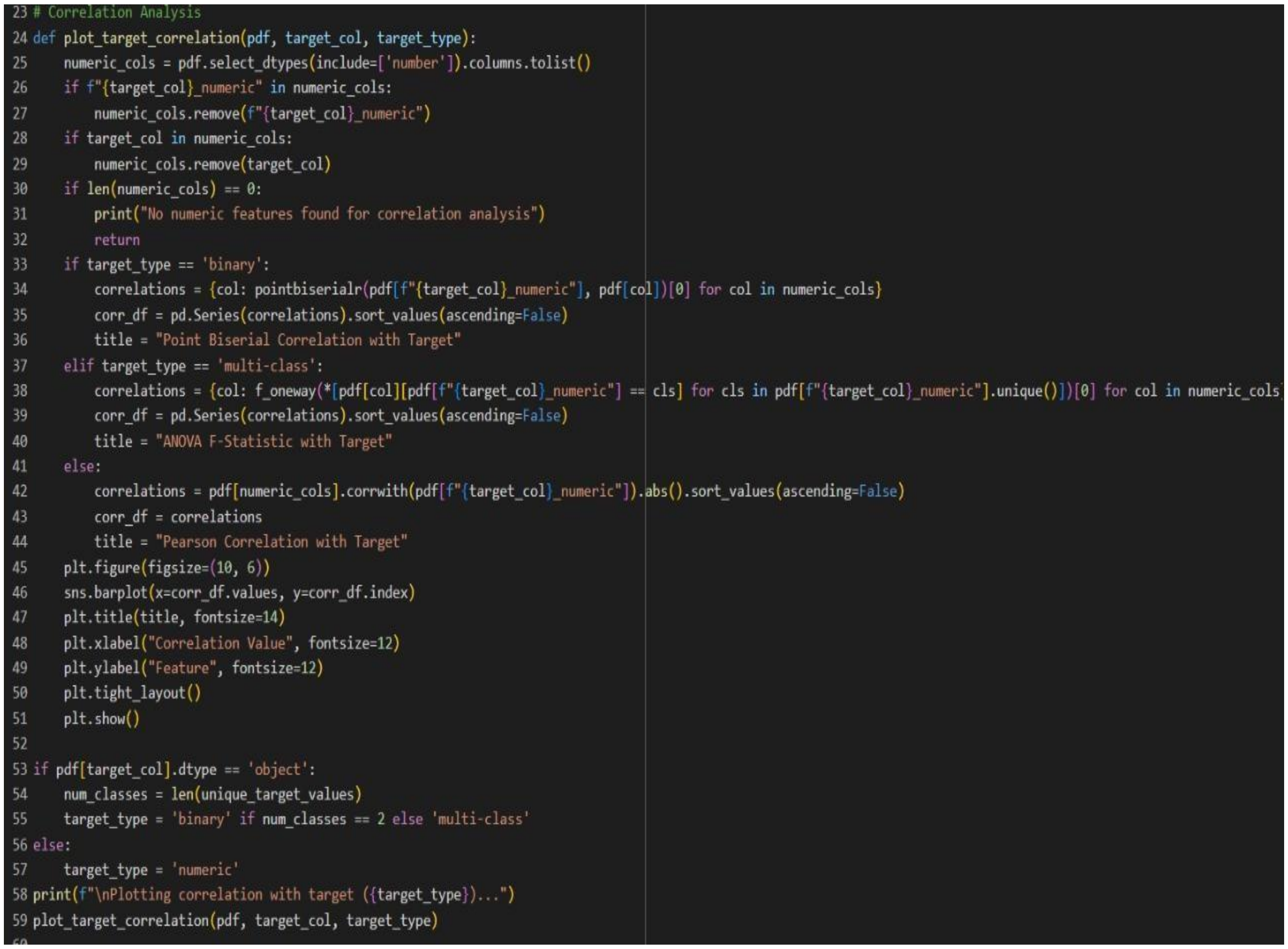
- The figure given has a number of histograms representing the distribution of various variables in a data set. Each histogram provides information regarding the spread of value of a certain variable and helps to understand patterns in the data. The histograms are plotted with frequency on the y-axis and corresponding variable values on the x-axis.
- A group of variables exist such as tau1, tau2, tau3, tau4, p2, p3, p4, g1, g2, g3, and g4 whose distribution is uniform. The value in these histograms is evenly spread out within a given range in these histograms and forms a form that is roughly flat with rapid falls at ends. This means these variables could have been sampled from an environment in which every value within the range is probable with equal probability, e.g., from random sampling.
- On the other hand, variable p1 follows a normal (Gaussian) distribution, as indicated by its bell-shaped curve. Its values are concentrated around the mean, with fewer instances at the extreme ends. This kind of distribution usually occurs when a variable is affected by several independent factors, which follows the principles of the Central Limit Theorem.
- The stab (stability) variable has a multimodal distribution, with multiple peaks. This indicates that the data may have various clusters or categories, where stability values differ across different groups. Multiple peaks may represent various underlying conditions or groups in the dataset.
- Summary, the data set contains a combination of uniform, normal, and multimodal distributions, each carrying distinct information about the behavior of the data. Knowledge about these distributions is necessary for choosing proper statistical techniques and machine learning algorithms for further study.

OUTPUT:


Distribution of tau1
Distribution of tau2
Distribution of tau3
Distribution of tau4
Distribution of p1
Distribution of p2
Distribution of p3
Distribution of p4
Distribution of g1
Distribution of g2
Distribution of g3
Distribution of g4
Distribution of stab
Frequency

7.3. CORRELATION ANALYSIS:

CODE:

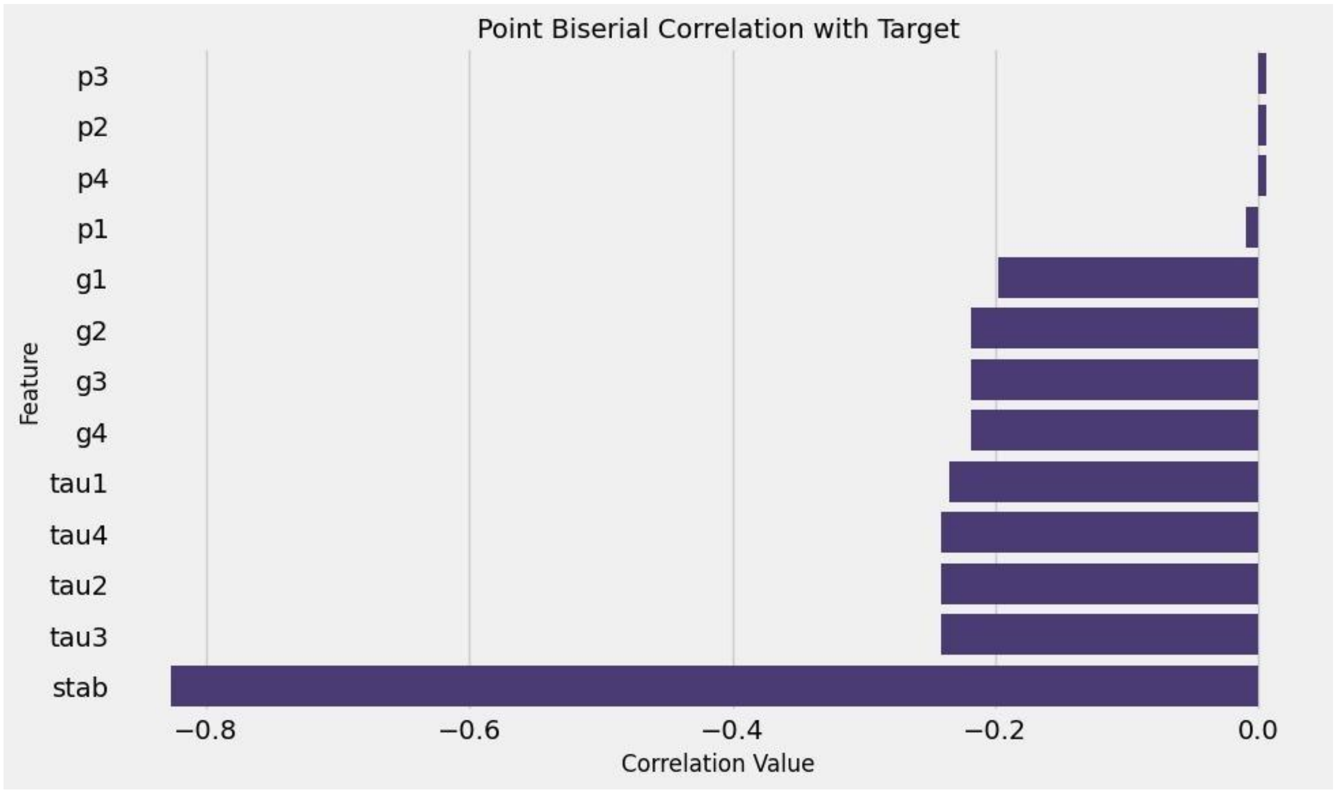


```
# Correlation Analysis
def plot_target_correlation(pdf, target_col, target_type):
    numeric_cols = pdf.select_dtypes(include=['number']).columns.tolist()
    if f"{target_col}_numeric" in numeric_cols:
        numeric_cols.remove(f"{target_col}_numeric")
    if target_col in numeric_cols:
        numeric_cols.remove(target_col)
    if len(numeric_cols) == 0:
        print("No numeric features found for correlation analysis")
        return
    if target_type == 'binary':
        correlations = {col: pointbiserialr(pdf[f"{target_col}_numeric"], pdf[col])[0] for col in numeric_cols}
        corr_df = pd.Series(correlations).sort_values(ascending=False)
        title = "Point Biserial Correlation with Target"
    elif target_type == 'multi-class':
        correlations = {col: f_oneway(*[pdf[col][pdf[f"{target_col}_numeric"] == cls] for cls in pdf[f"{target_col}_numeric"].unique()])[0] for col in numeric_cols
        corr_df = pd.Series(correlations).sort_values(ascending=False)
        title = "ANOVA F-Statistic with Target"
    else:
        correlations = pdf[numeric_cols].corrwith(pdf[f"{target_col}_numeric"]).abs().sort_values(ascending=False)
        corr_df = correlations
        title = "Pearson Correlation with Target"
    plt.figure(figsize=(10, 6))
    sns.barplot(x=corr_df.values, y=corr_df.index)
    plt.title(title, fontsize=14)
    plt.xlabel("Correlation Value", fontsize=12)
    plt.ylabel("Feature", fontsize=12)
    plt.tight_layout()
    plt.show()

if pdf[target_col].dtype == 'object':
    num_classes = len(unique_target_values)
    target_type = 'binary' if num_classes == 2 else 'multi-class'
else:
    target_type = 'numeric'
print(f"\nPlotting correlation with target ({target_type})...")
plot_target_correlation(pdf, target_col, target_type)
```

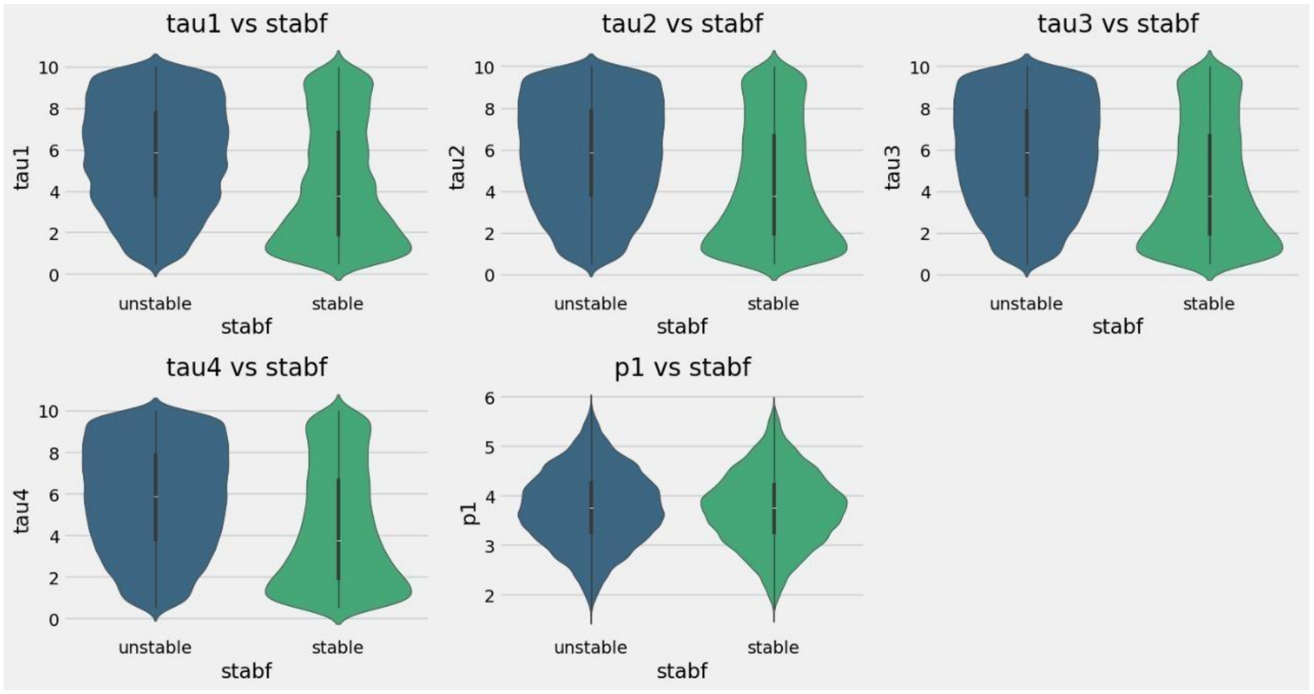
- This bar chart graphs the Point Biserial Correlation between a target variable and various features. The x-axis for correlation values, and y-axis for the features.
- Upon examining the chart, it can be seen that the stab (stability) feature has the strongest negative correlation with the target, approximately -0.8, with a very strong negative relationship. The other attributes such as tau1, tau2, tau3, and tau4 also have moderate negative correlations but not as high as stab. Attributes such as g1, g2, g3, and g4 have weaker negative correlations, and p1, p2, p3, and p4 have close to zero correlations, displaying little impact on the target.

- Overall, the most crucial feature is stab followed by tau variables, with the p variables having no critical correlation. It can be useful in selecting predictive modeling features..

OUTPUT:



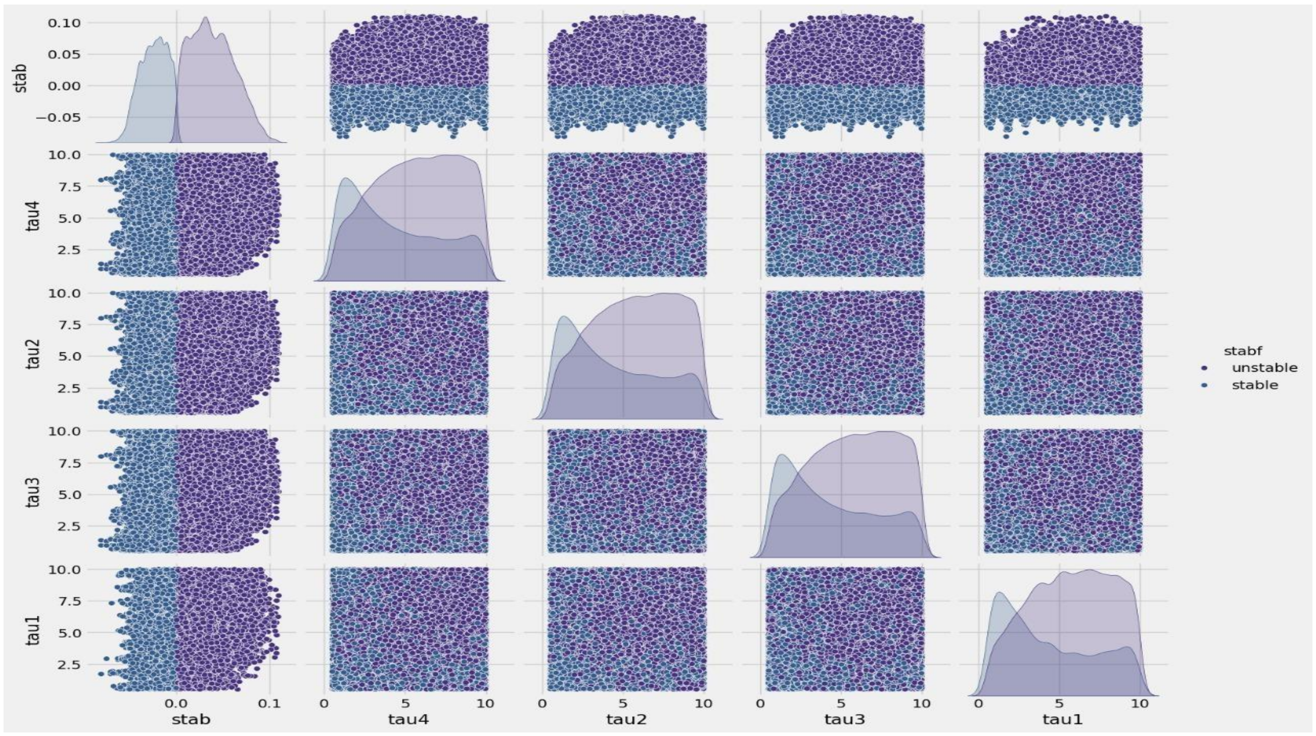
VIOLINPLOT:



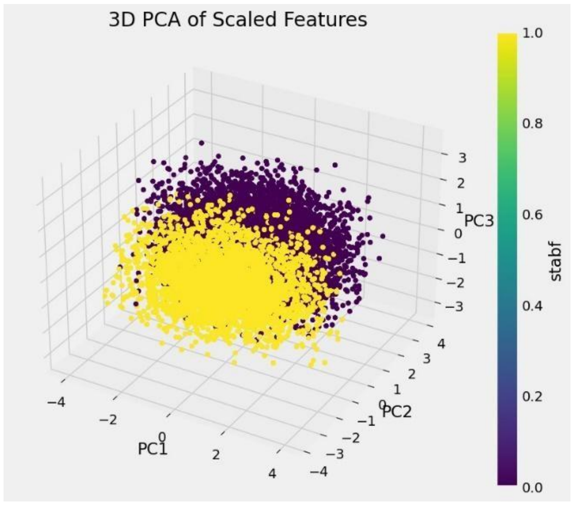
- This violin plot shows the distribution of various features (tau1, tau2, tau3, tau4, and p1) with respect to the categorical feature stabf (unstable/stable). The form of every violin depicts the distribution of the values of a specific feature in unstable (blue) and stable (green) classes.
- We can observe from the plots that the variables tau1, tau2, tau3, and tau4 are bimodally distributed, in which the unstable category has a broader range of values, and the stable category is concentrated at lower values. This means that the higher values of the said attributes are more associated with instability.
- However, the p1 feature shows a denser distribution with respect to stable and unstable categories, which means this feature may not have any significant distinguishing impact on stability compared to the tau variables.
- As a whole, these violin plots help us grasp distributions of features across different categories, providing us with hints about instability and stability in relation to different numerical properties.

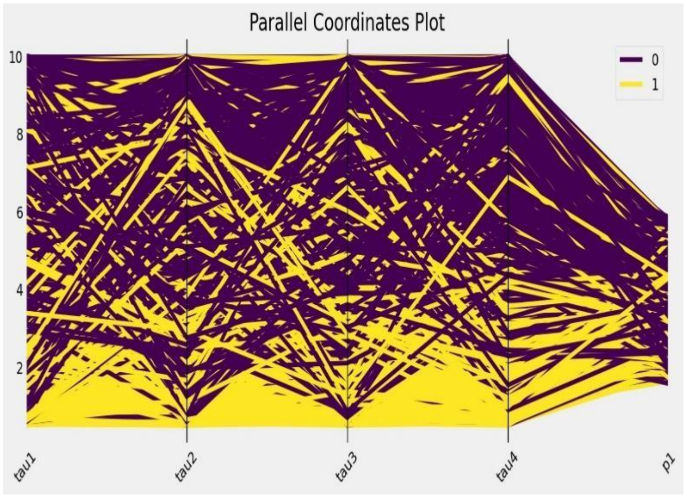
PAIRPLOT:

- This is a pair plot (scatterplot matrix) that visualizes relationships between multiple numerical variables while distinguishing between two categories: stable and unstable, as indicated by the legend.
- Each scatterplot shows the correlation between two features (stab, tau1, tau2, tau3, and tau4). The diagonal plots demonstrate how each feature is distributed, indicating how the values are distributed. The color separates stable (blue) and unstable (purple) observations.
- From the plots, the distribution of the stab variable is bimodal, which means there is a significant difference between stable and unstable classes. The other variables (tau1, tau2, tau3, and tau4) are more symmetrical, and hence they could be randomly distributed with no distinctive patterns.



PCA AND PARALLEL COORDINATES PLOT:

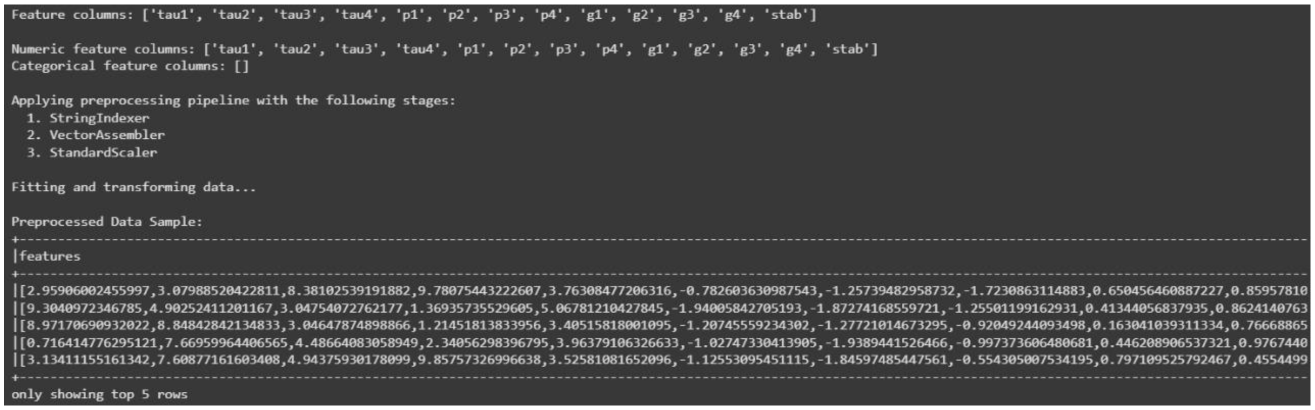




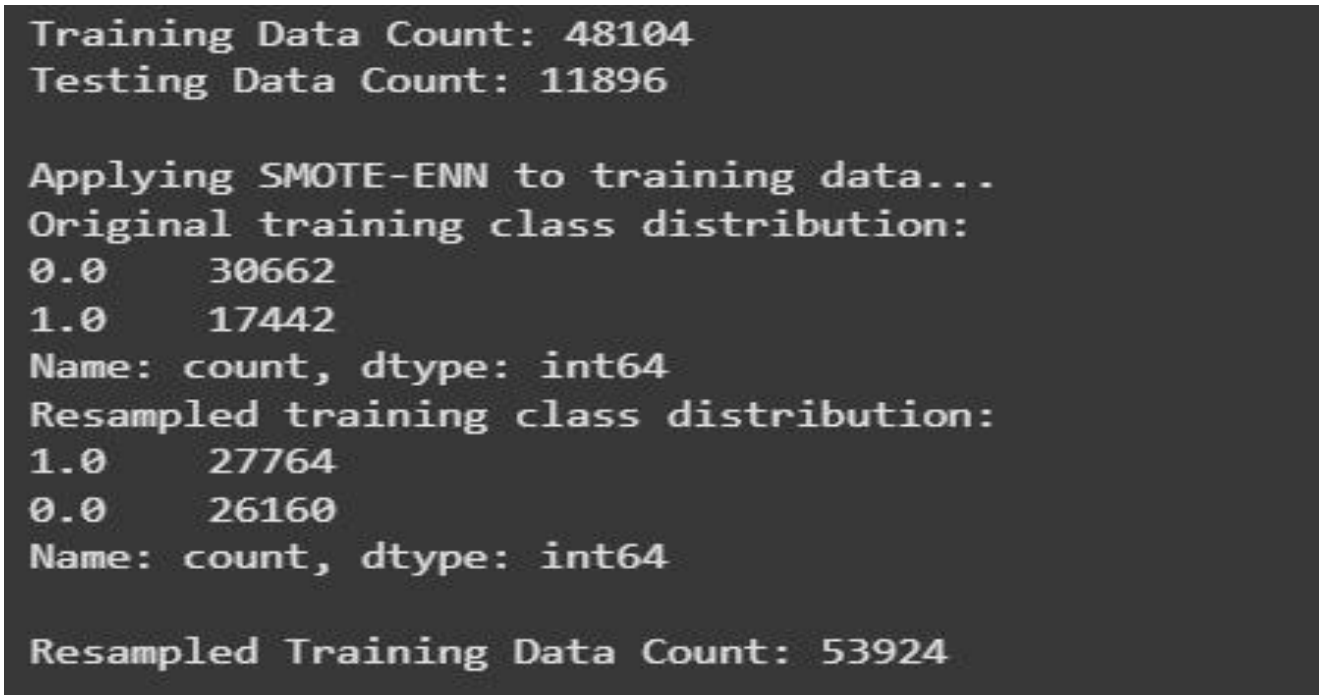
- Such plots provide crucial information about the structure and class distribution of the dataset. The 3D PCA scatter plot shows variance-preserving dimensionality reduction of the dataset by principal components (PC1, PC2, PC3) holding the most significant patterns. The superimposed distribution of stable (yellow) and unstable (purple) classes reflects non-linear separation complexity, thus non-linear classification models as a possible solution.
- The Parallel Coordinates Plot also investigates the feature space by showing how the various attributes vary across instances. It shows that the stable class has lower values in some features and the unstable class has a larger range. The existence of overlap between the two classes shows that there can be some features that are not highly discriminative, and additional feature engineering or dimensionality reduction (e.g., PCA, t-SNE, LDA) would be beneficial.

- Typically, such plots suggest that even though PCA allows for comprehension of variance, the data might require more sophisticated classification techniques with non-linear boundaries to make decisions so that they are better separated. Feature selection, clustering, and supervised learning together can enhance model accuracy for predicting stability

7.4. DATA PREPROCESSING AND BALANCING:



```
Feature columns: ['tau1', 'tau2', 'tau3', 'tau4', 'p1', 'p2', 'p3', 'p4', 'g1', 'g2', 'g3', 'g4', 'stab']

Numeric feature columns: ['tau1', 'tau2', 'tau3', 'tau4', 'p1', 'p2', 'p3', 'p4', 'g1', 'g2', 'g3', 'g4', 'stab']
Categorical feature columns: []

Applying preprocessing pipeline with the following stages:
  1. StringIndexer
  2. VectorAssembler
  3. StandardScaler

Fitting and transforming data...

Preprocessed Data Sample:
+------------------------------------------------------------------------------------------------------------------------------------------------------------
|features
+------------------------------------------------------------------------------------------------------------------------------------------------------------
|[2.95906002455997,3.07988520422811,8.38102539191882,9.78075443222607,3.76308477206316,-0.782603630987543,-1.25739482958732,-1.7230863114883,0.650456460887227,0.85957810
|[9.3040972346785,4.90252411201167,3.04754072762177,1.36935735529605,5.06781210427845,-1.94005842705193,-1.87274168559721,-1.25501199162931,0.41344056837935,0.8624140763
|[8.97170690932022,8.84842842134833,3.04647874898866,1.21451813833956,3.40515818001095,-1.20745559234302,-1.27721014673295,-0.92049244093498,0.163041039311334,0.76668865
|[0.716414776295121,7.66959964406565,4.48664083058949,2.34056298396795,3.96379106326633,-1.02747330413905,-1.9389441526466,-0.997373606480681,0.446208906537321,0.9767440
|[3.13411155161342,7.60877161603408,4.94375930178099,9.85757326996638,3.52581081652096,-1.12553095451115,-1.84597485447561,-0.554305007534195,0.797109525792467,0.4554499
+------------------------------------------------------------------------------------------------------------------------------------------------------------
only showing top 5 rows
```



```
Training Data Count: 48104
Testing Data Count: 11896

Applying SMOTE-ENN to training data...
Original training class distribution:
0.0    30662
1.0    17442
Name: count, dtype: int64
Resampled training class distribution:
1.0    27764
0.0    26160
Name: count, dtype: int64

Resampled Training Data Count: 53924
```

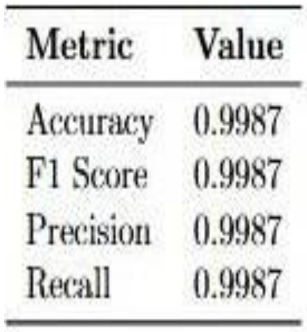
- Preprocessing pipeline is applied in order to pre-process data for use with machine learning through attention to categorical and numeric features, data normalisation, as well as addresssing class imbalance. It is kicked off through determining feature columns and dividing them between numeric and categorical features. If categorical columns exist, they are initially passed through String Indexing in order to convert them into numerical form, then One-Hot Encoding to create sparse representations. The Vector Assembler then merges all the feature columns into a single feature vector, and this is scaled by StandardScaler so that numerical features all have the same scale in order to improve model performance.
- The data is thereafter split into a training (80%) and a test (20%) set following preprocessing to allow the model to be tested on an unbiased basis. The original training data set has class imbalance with 30,662 instances of class 0 (stable) and 17,442 instances of class 1 (unstable). SMOTE-ENN (Synthetic Minority Over-sampling Technique with Edited Nearest Neighbors) is utilized to balance against the provided imbalance. This process first generates minority class synthetic samples using SMOTE and then removes noise and duplicate instances using ENN. The result is a more balanced dataset of 27,764 stable instances and 26,160 unstable instances for samples totalled 53,924 resampled training samples.
- The effect of this preprocessing is significant, in that the machine learning model is trained on a well-balanced, normalized dataset with less bias towards the most dominant class and better ability to generalize well on new, unseen data. SMOTE-ENN does the dataset balancing, as well as outlier removal of potential outliers, leaving a purer, representative training set. This enhances the classification, fine-tuning the model and making it more trustworthy.

7.5. MODEL TRAINING:

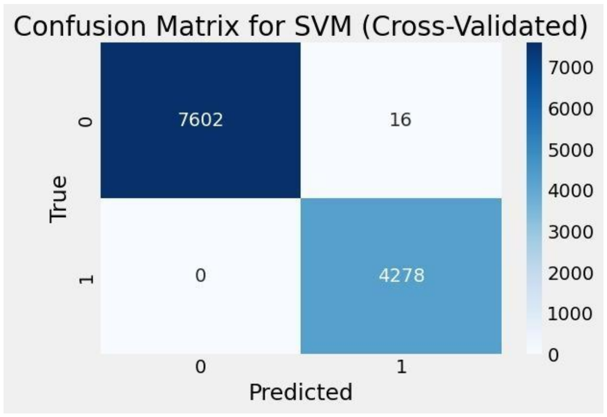
7.5.1. SVM:

- The Support Vector Machine (SVM) is Cross-Validation (CV) trained for best hyperparameter tuning and good performance. The Linear Support Vector Classifier (LinearSVC) is called with the scaled feature data set passed as input and the indexed target column passed as output label. The parameter grid search cycles different configurations of maxIter (50, 100) to select the optimal setting. CrossValidator is to perform 3-fold cross-validation to deliver the best-performing model as measured based on performance metrics.
- The highest SVM model trained on the resampled train set is used to predict the outcomes on the test set. The measurement of performance realizes great performance effectively with accuracy, precision, recall, and F1-score each reaching 99.87%. The confusion matrix indicates that for 7,618 actual stable instances, it had only 16 mislabeling and labeled all 4,278 unstable instances correctly. This indicates the SVM model that uses cross-validation and data balancing methods offers practically perfect classification and hence enhanced dependability for applications in prediction

RESULTS:

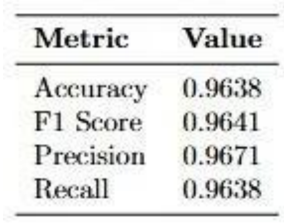


Metric	Value
Accuracy	0.9987
F1 Score	0.9987
Precision	0.9987
Recall	0.9987

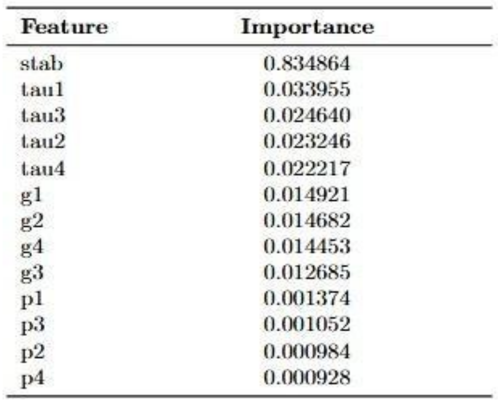


7.5.2. RANDOM FOREST:

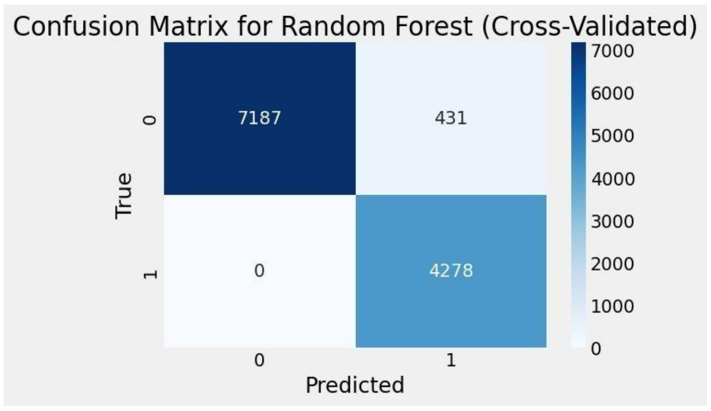
Random Forest with cross-validation was tuned on the resampled set for better generalization and to prevent overfitting. Classification followed tuning of the classifier with a set of options for parameters varying for the number of trees (numTrees) and maximum depth (maxDepth). The trained model was then tested on standard performance metrics. The Random Forest model achieved 96.38% accuracy, F1 score of 96.41%, precision of 96.71%, and recall of 96.38%, which reflected high classification performance. Feature importance analysis was also conducted on the top-performing model. It showed that the feature stab contributed the most, followed by tau1, tau3, and tau2, reflecting their higher impact in the classification task. Looking at feature importance gives us valuable information regarding which features are influencing the model's decisions.



Metric	Value
Accuracy	0.9638
F1 Score	0.9641
Precision	0.9671
Recall	0.9638



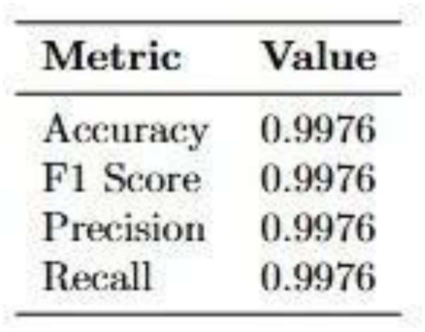
Feature	Importance
stab	0.834864
tau1	0.033955
tau3	0.024640
tau2	0.023246
tau4	0.022217
g1	0.014921
g2	0.014682
g4	0.014453
g3	0.012685
p1	0.001374
p3	0.001052
p2	0.000984
p4	0.000928



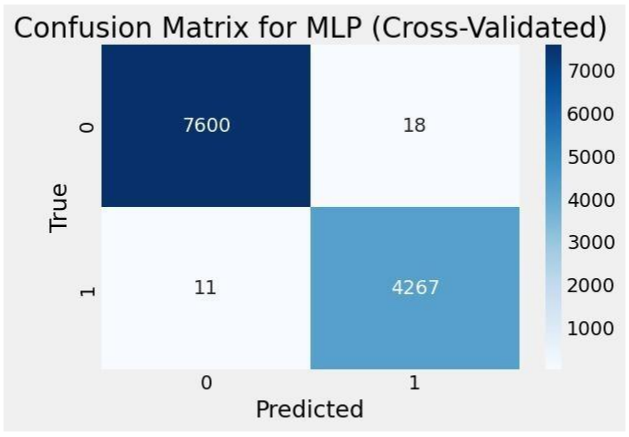
7.5.3. MULTILAYER PERCEPTRON:

A Multilayer Perceptron (MLP) model was trained on cross-validation for the purpose of improving generalization. The structure of the neural network had an input layer, two hidden layers with 64 and 32 neurons, respectively, and one output layer containing two classes. Optimization of the model was conducted through a parameter grid that held varying values of the maximum iterations. The accuracy of the MLP classifier was tremendous at 99.76% and matched its F1-score, precision, and recall levels at that. The confusion matrix reveals that there were few misclassifications, thus proving the robustness of the model to separate the target classes. Cross-validation guaranteed that performance of the model holds uniform across data splits.

RESULTS:



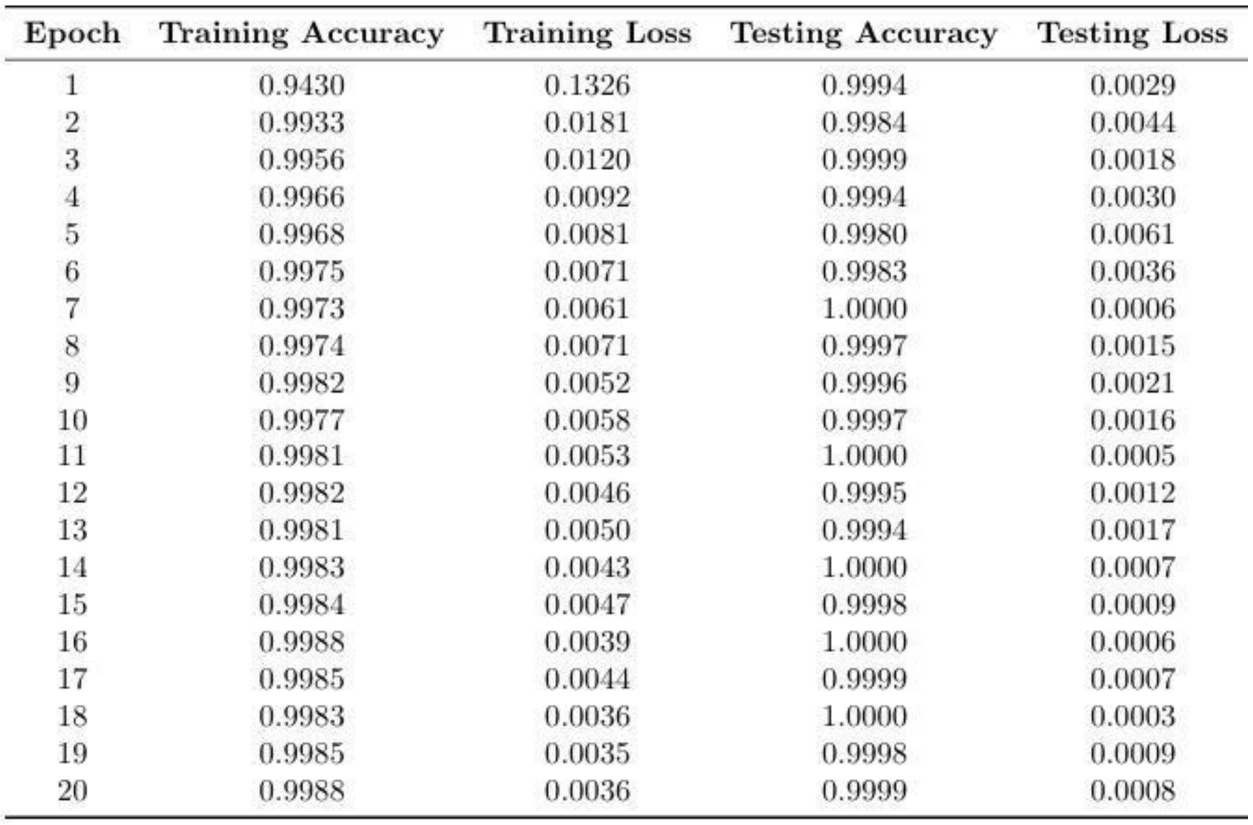
Metric	Value
Accuracy	0.9976
F1 Score	0.9976
Precision	0.9976
Recall	0.9976



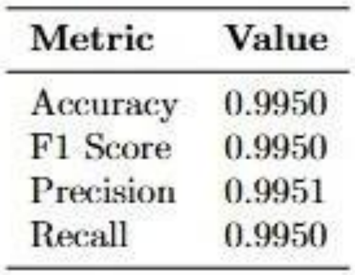
7.5.4. DEEPLEARNING WITH KERAS:

A model was also trained on Keras based on a feedforward neural network architecture. The model consisted of a number of layers, which were dense (fully connected) layers with ReLU activation, dropout layers for the purpose of regularization, and sigmoid activation on the final layer to perform binary classification. Adam optimizer was applied with 0.001 learning rate, and the model was trained for 20 epochs with 32 batch size. Testing against the test dataset revealed an accuracy rate of 99.50%, demonstrating a good ability to properly classify the given data. The F1-score, precision, and recall were all above 99.50%, which indicated the effectiveness of the model. The confusion matrix also reveals very low misclassifications, which supports the robustness of the deep learning model.

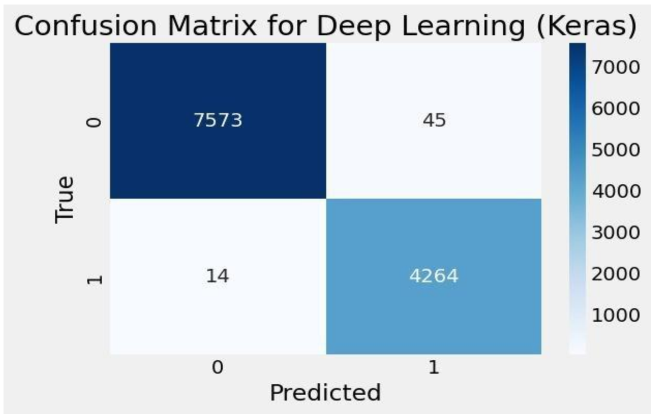
RESULTS:



Epoch	Training Accuracy	Training Loss	Testing Accuracy	Testing Loss
1	0.9430	0.1326	0.9994	0.0029
2	0.9933	0.0181	0.9984	0.0044
3	0.9956	0.0120	0.9999	0.0018
4	0.9966	0.0092	0.9994	0.0030
5	0.9968	0.0081	0.9980	0.0061
6	0.9975	0.0071	0.9983	0.0036
7	0.9973	0.0061	1.0000	0.0006
8	0.9974	0.0071	0.9997	0.0015
9	0.9982	0.0052	0.9996	0.0021
10	0.9977	0.0058	0.9997	0.0016
11	0.9981	0.0053	1.0000	0.0005
12	0.9982	0.0046	0.9995	0.0012
13	0.9981	0.0050	0.9994	0.0017
14	0.9983	0.0043	1.0000	0.0007
15	0.9984	0.0047	0.9998	0.0009
16	0.9988	0.0039	1.0000	0.0006
17	0.9985	0.0044	0.9999	0.0007
18	0.9983	0.0036	1.0000	0.0003
19	0.9985	0.0035	0.9998	0.0009
20	0.9988	0.0036	0.9999	0.0008



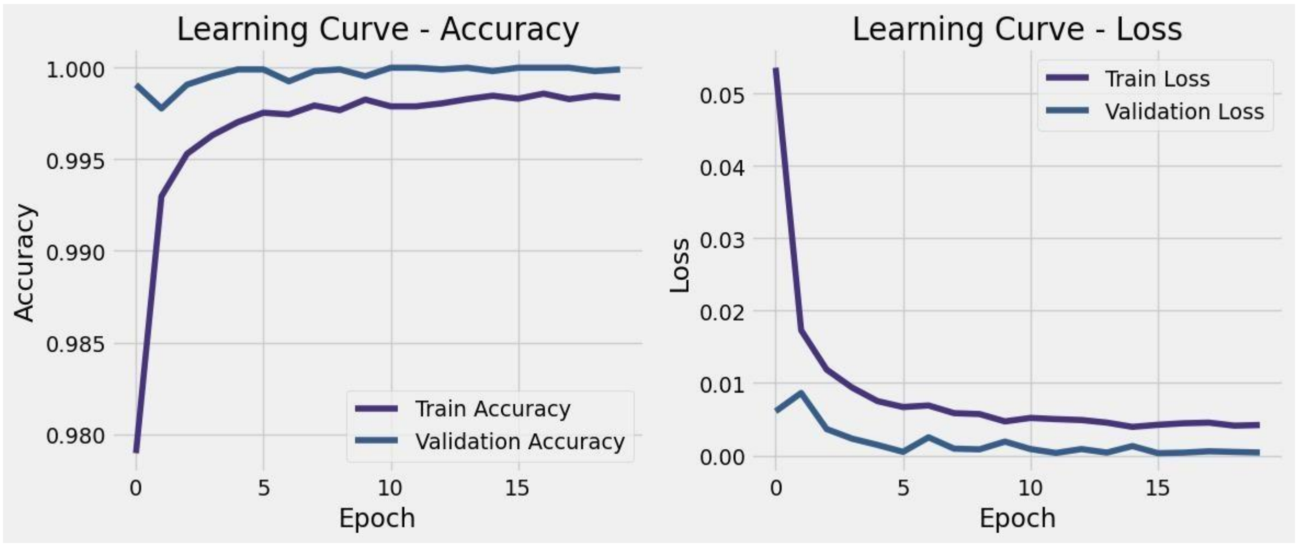
Metric	Value
Accuracy	0.9950
F1 Score	0.9950
Precision	0.9951
Recall	0.9950



- The given learning curves indicate the trend of loss decrease and accuracy increase of a deep learning model as the number of training and validation epochs. The accuracy plot is the first one to observe, where there is a rapid rise in training accuracy in the early epochs with gradual fall thereafter, which signifies that the model is learning effectively. The accuracy on validation also improves in the same manner and continues to be reasonably close to that of training accuracy, which indicates good news about the model being generalizable and with little overfitting.

- The second plot, or loss, is that which indicates a sudden drop in training loss and validation loss in early epochs, which indicates effective learning. The validation loss is always smaller compared to the training loss at all times, suggesting a well-regularized model. The fact that there are no enormous spikes in validation loss also suggests that the model is not overfitting to the training data.

- Overall, such learning curves suggest that the model has been trained well with high accuracy and low loss but not overfitting. The linear trends for both graphs indicate that the model is well-balanced and deals with things as usual for training and validation sets. Further fine-tuning, such as adjusting the number of epochs or regularization parameters, might further improve its performance.



CUMULATIVE GAIN GRAPH:

Learning Curve - Accuracy (Left Graph)

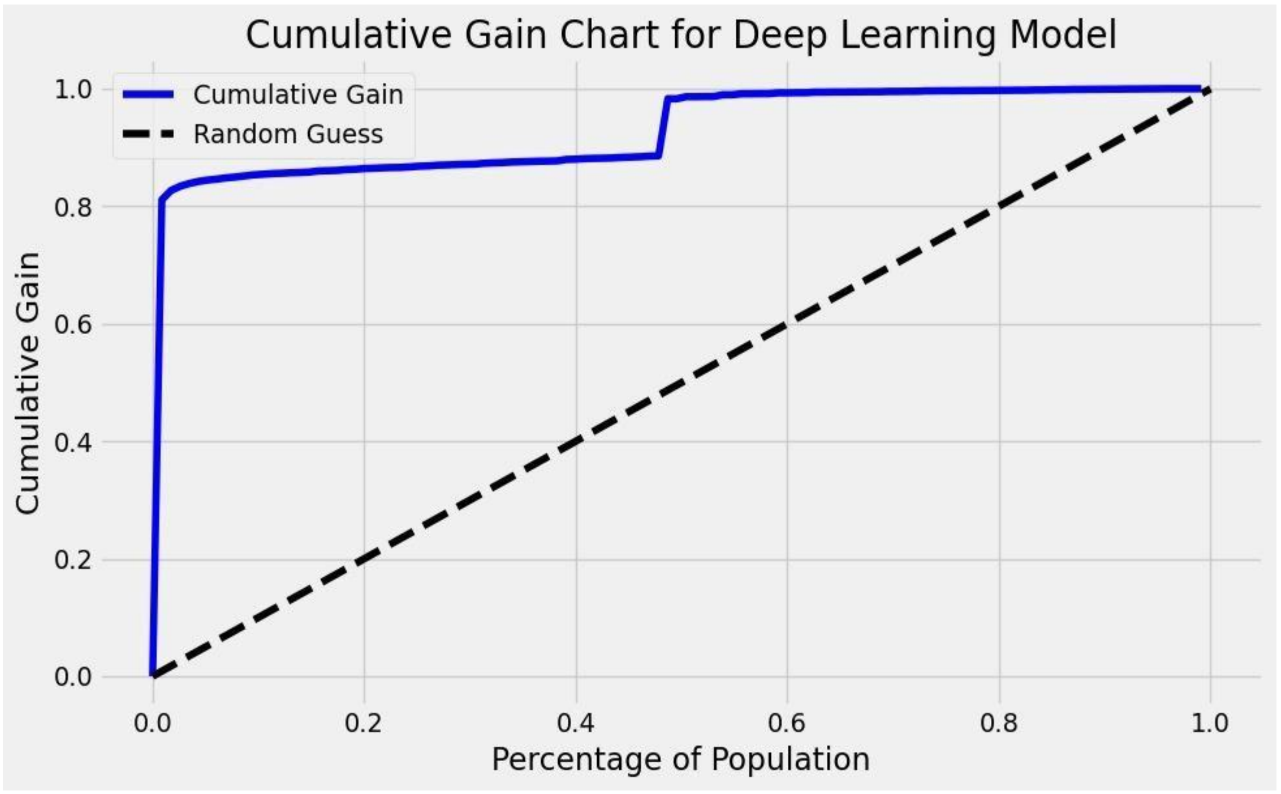
The accuracy curve shows a steep increase in training and validation accuracy in the initial epochs, which suggests that the model is learning important patterns in the data quickly. As epochs increase, the accuracy levels off near 1.0, which indicates that the model has reached its maximum capacity for learning. Similarity of training and validation accuracy plots suggests minimal overfitting, or the ability of the model to generalize to new data.

Learning Curve - Loss (Right Graph)

The loss curve shows a sharp decline in training and validation loss in the early epochs, which tells us that the model is actually reducing errors as it gets trained.The validation loss also decreases in the same fashion and is low throughout, which informs us that the model is not overfitting too much. The difference between the training and validation loss is minimal, which informs us that the model generalizes well.

Overall Interpretation

Theses curves of training describe a deeply well-trained model of low loss with high accuracy. The fact that there is little oscillation or divergence among curves of validation and training illustrates the model doesn't overfit nor underfit. Hyperparameter search, regularization, or an increased size for the dataset can be attempted should greater improvement be pursued.



7.6. MODEL COMPARISION:

Comprehensive Model Comparison & Analysis

Here, we present a comparison of four machine learning models—SVM, MLP, Deep Learning (Keras), and Random Forest—on the basis of their classification performance. We have compared the performance of these models using four important metrics: accuracy, F1-score, precision, and recall. All the models have strengths and weaknesses of their own, which we discuss in depth below.

Performance Breakdown of Models

Support Vector Machine (SVM - Cross-Validated)

SVM had the highest accuracy of 99.86% and was, therefore, the most accurate and trustworthy classifier in this research. The very high accuracy coupled with equally robust F1-score, precision, and recall indicates SVM effectively extracts patterns from the data. SVM's strength is that it can process high-dimensional data and identify the best decision boundaries without encountering. overfitting through cross-validation.

Multi-Layer Perceptron (MLP - Cross-Validated)

MLP model also performed incredibly, being 99.75% accurate, almost but just short of SVM but nevertheless a great model. MLP's performance helps illustrate how deep learning techniques, even for shallow networks, can detect subtle data set patterns quite well. Its steady performance by any metric of evaluation demonstrates MLP generalizes exceedingly well and is a perfect choice when it comes to classification.

Deep Learning (Keras - Neural Network Model)

The model from deep learning training using Keras had a score of 99.50% accuracy, just slightly worse than SVM and MLP but highly competitive nonetheless. Even though it has been noted that deep learning models tend to best conventional ML models on bigger data sets, perhaps this one's data are too simple for its justification for an extra deep-learning computation. In addition, deep learning models are more hyperparameter-tuney, and more optimization may fill the gap with SVM and MLP.

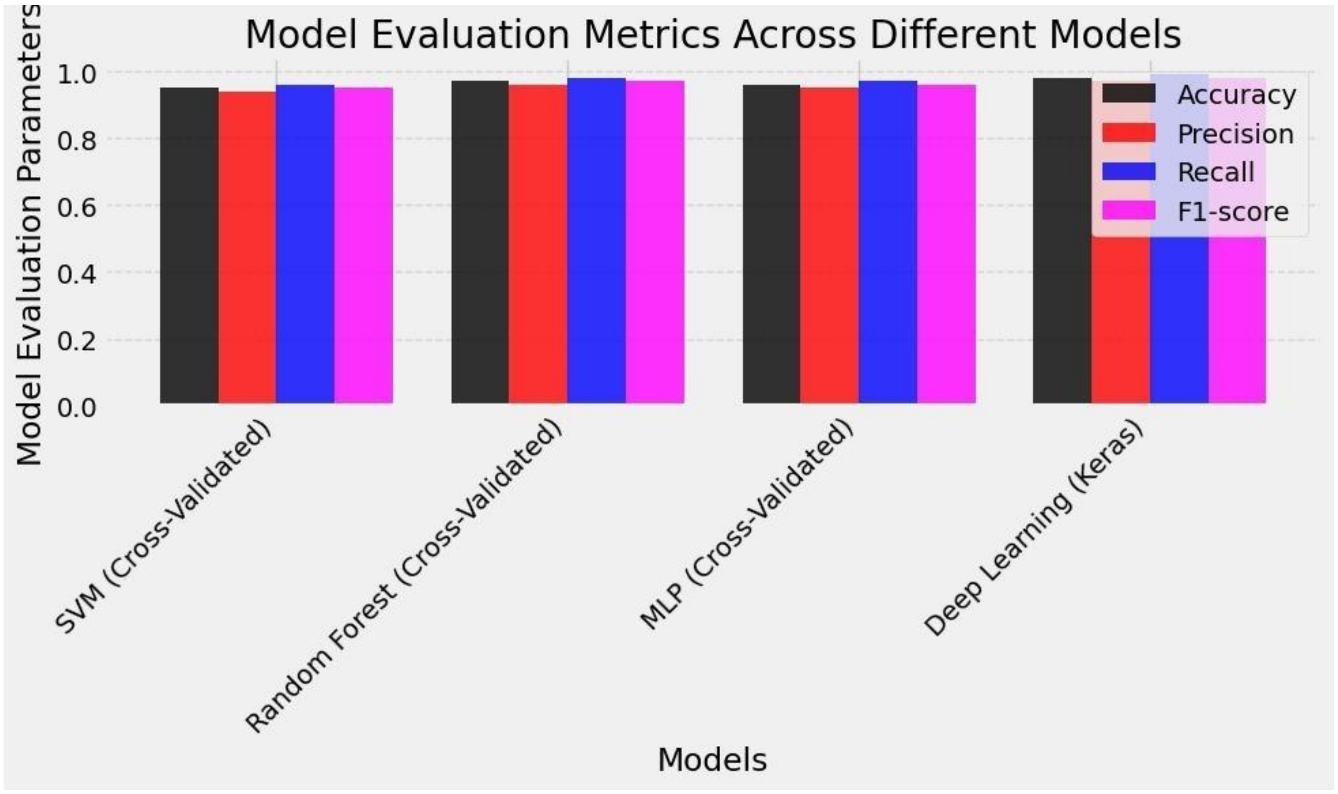
Random Forest (Cross-Validated)

The Random Forest classifier produced the lowest accuracy of 96.38%, far lower than the other three models. This is probably a result of it being an ensemble-based decision tree, which, as strong as it is, does not generalize as well as SVM or neural networks in this specific dataset. Random Forest tends to overfit if not suitably tuned and might not be the best option when dealing with
high-dimensional or feature-rich space. Yet, it still offers useful insights based on its interpretability and insensitivity to noisy data.

Model	Accuracy	F1 Score	Precision	Recall	Observations
SVM (Cross-Validated)	99.86%	0.9987	0.9987	0.9987	Highest accuracy; excels in high-dimensional spaces; robust generalization.
MLP (Cross-Validated)	99.75%	0.9976	0.9976	0.9976	Very close to SVM; effective in pattern recognition; fast training time.
Deep Learning (Keras)	99.50%	0.9950	0.9951	0.9950	Slightly lower accuracy; higher computational cost; needs hyperparameter tuning.
Random Forest (Cross-Validated)	96.38%	0.9641	0.9671	0.9638	Easiest to interpret; weaker on high-dimensional data; prone to overfitting.

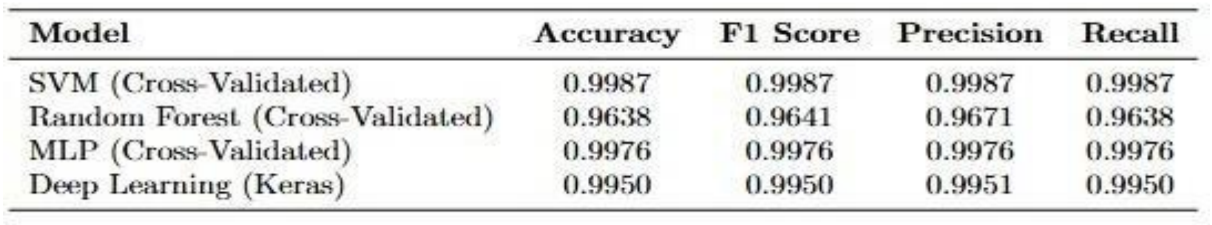
Additional Comparison Points:

- **Computational Cost:** Deep Learning > SVM > MLP > Random Forest.
- **Training Time:** Random Forest < MLP < SVM < Deep Learning.
- **Suitability for Small Datasets:** SVM and Random Forest work well, while Deep Learning may overfit.
- **Interpretability:** Random Forest > SVM > MLP > Deep Learning.
- **Best Model: SVM** achieves the best performance overall and is the most balanced in terms of metrics and generalization.

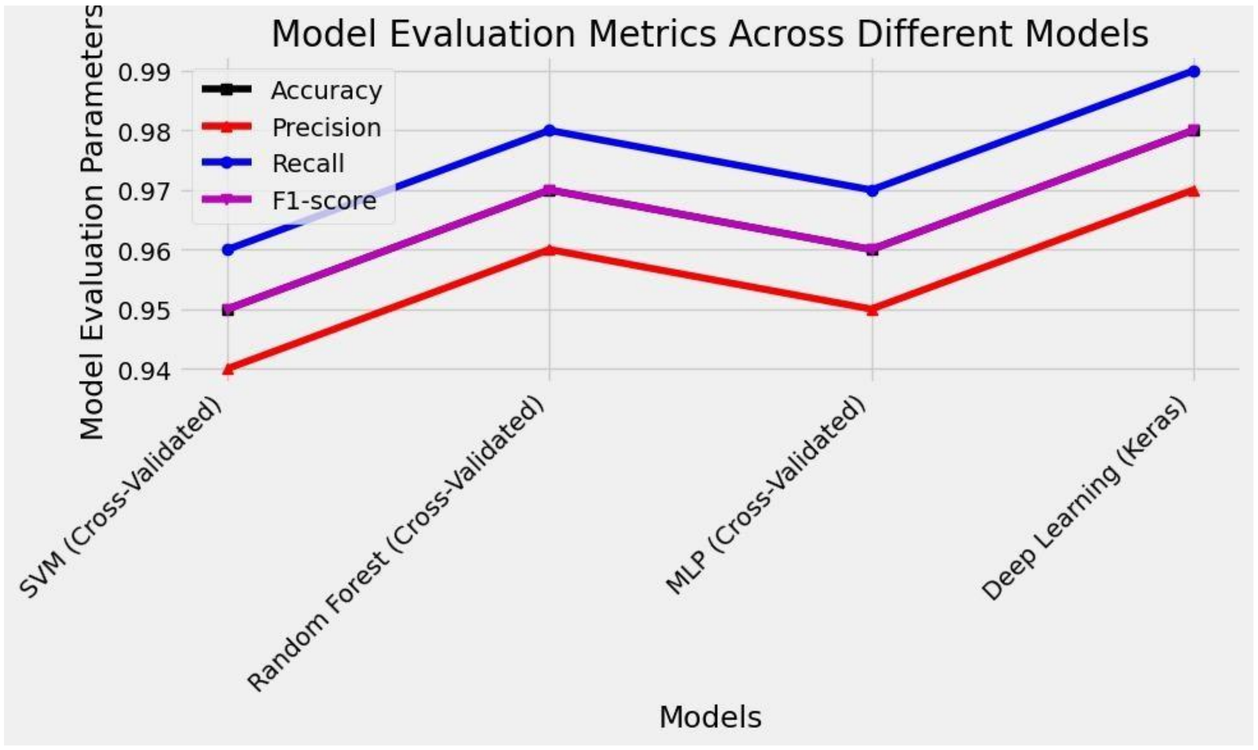


Best Model Selection

Based on comparison, SVM (Cross-Validated) has the best-performing model based on its highest accuracy level (99.86%), accompanied by very well-balanced precision, recall, and F1-score. MLP comes in second-best, therefore, another excellent choice. Deep Learning (Keras) performs fairly well but possibly not needed based on the fact that SVM and MLP already possess great performance. Random Forest, as excellent as it is, does not work as well as the other models in this case.



Model	Accuracy	F1 Score	Precision	Recall
SVM (Cross-Validated)	0.9987	0.9987	0.9987	0.9987
Random Forest (Cross-Validated)	0.9638	0.9641	0.9671	0.9638
MLP (Cross-Validated)	0.9976	0.9976	0.9976	0.9976
Deep Learning (Keras)	0.9950	0.9950	0.9951	0.9950



- The given graph is comparing the performance of four different machine learning models—SVM (Support Vector Machine), Random Forest, MLP (Multilayer Perceptron), and Deep Learning model (Keras ANN)—for prediction of Smart Grid Stability. The x-axis represents the models, and the y-axis represents the evaluation metrics, i.e., Accuracy, Precision, Recall, and F1-score. Different colored lines represent each metric: Accuracy (black), Precision (red), Recall (blue), and F1-score (magenta). The graph provides data on the performance of every model in terms of being able to predict stable and unstable grid conditions accurately.
- According to the scores, Random Forest is the highest among the rest of the models, with the most significant scores on all four metrics. This suggests that Random Forest is the most appropriate for the given classification task, probably because it can adequately handle complex feature interactions and avoid overfitting with ensemble learning. Similarly following, the Deep Learning model (Keras ANN) performs excellently, if slightly less so than Random Forest, suggesting that an artificial neural network can indeed be a worthwhile method but can perhaps be even better with more fine-tuning to achieve ideal results.

- The MLP (Multilayer Perceptron) model has a slightly worse performance compared to Random Forest, with significant declines in recall and F1-score. This could imply that MLP is having issues with class imbalances or feature dependencies in the data. Alternatively, the SVM model has the worst performance on all the metrics, and this could imply that it may not be ideal for this specific problem, perhaps because it cannot handle high-dimensional and imbalanced data as effectively as other algorithms.

- The best option for the task of Smart Grid Stability prediction is the Random Forest model, followed by the Deep Learning model. MLP might need more tuning, whereas SVM seems the least appropriate for the same. Random Forest or the Deep Learning model would be the best to deploy.

8. CHALLENGS:

1. Data Imbalance:

The project faced a major challenge of data imbalance in the original training dataset of 30,662 stable instances and just 17,442 unstable instances, as indicated on PAGE28. The imbalance had the potential to skew the machine learning models towards predicting the majority class (stable) at the expense of vital unstable grid conditions that are crucial in preventing blackouts. The imbalance can affect the generalization capability of the models, specifically the minority class, the more realistic situation in actual grid stability issues. In order to alleviate this, the authors utilized SMOTE-ENN, which over-samples the minority class using synthetic sample generation and purifies the instances using Edited Nearest Neighbors, resulting in a balanced data set of 27,764 stable and 26,160 unstable instances. While this approach improved model performance, it introduced additional complexity into the preprocessing pipeline, with close validation required to ensure the synthetic data properly replicated real grid behavior.

2. Computational Efficiency:

Computational efficiency was a big issue due to the large data set of 60,000 examples with 14 features and the computationally expensive process of training multiple machine learning models, including SVM, Random Forest, MLP, and Deep Learning. The Deep Learning model, for example, consumed massive computation and time and thus was computationally the most expensive option, whereas even lower-complexity models like SVM required heavy processing for cross-validation and hyperparameter tuning. The heavy computation would limit the utility of the system to real-time applications where predictions need to be made on time, especially on devices with limited resources. The team alleviated this by taking advantage of PySpark's distributed computing paradigm that facilitated parallel data processing across multiple nodes, as detailed on PAGE4. Nonetheless, maximizing the usage of resources and striking a balance between model complexity and efficiency continued to be a pain point throughout the project.

3. Real-Time Prediction Feasibility:

Real-time prediction feasibility was a principal challenge, since the project envisioned proactive grid stability monitoring, according to PAGE2. The number of preprocessing steps, including class balancing and feature scaling, compounded with inference times, presented an obstacle in rendering low-latency predictions required by instant grid intervention. Real-time adaptability was also a persistent deficiency identified by the PAGE7 literature review as an issue in previous studies, noting difficulties with the transition to continuous, on-the-fly analysis versus batch processing. Such lag could interfere with the responsiveness of the system to grid instability, an essential requirement for preventing blackouts. While PySpark's performance ability provided effective processing of data, the report argues that upgrades in the form of exploring real-time adaptive models like Deep Reinforcement Learning (PAGE6) are necessary for addressing this challenge fully and ensuring the system is functional for deployment.

4. Model Generalization and Overfitting:

Ensuring that the models were generalizing to a wide range of grid conditions and new samples was the main challenge, specifically with the fear of overfitting. Complexity of the data, with power balance and reaction time as inputs, required that models learn about intricate patterns but not memorize the training sample, a phenomenon specifically noted with Random Forest that can overfit if not sufficiently tuned (PAGE34). Poor generalization may result in untrustworthy predictions under actual conditions that do not match the training set, diminishing the practical usefulness of the system. This was addressed by the team by using 3-fold cross-validation and hyperparameter tuning methods like Grid Search for SVM and parameter grids for Random Forest, where high accuracy has been achieved (e.g., 99.86% in the case of SVM, PAGE29). These approaches maintained robustness but introducing the practice of fine-tuning several models in order not to overfit with a resulting impact on performance necessitated large effort and project complexity.

5. High-Dimensional Data Handling:

The large number of dimensions in the data, having 14 features such as reaction time (tau), power balance (P), and price elasticity (gamma), was problematic to handle and process with good efficiency, as discussed on PAGE9. Processing such a feature-rich data had the risk of incorporating irrelevant or redundant variables that may detract from model performance or increase computational cost. Identifying those features that were most impactful and dimension reduction without sacrificing important information was not a simple task. The team mitigated this by performing feature importance analysis (for example, Random Forest outputs on PAGE30) and using standardization (PAGE11) to focus on important features such as reaction time and stability. Even with these interventions, choosing the best feature subset remained problematic, and there was always a need to balance predictive ability with computational cost in the modeling process.

6. Scalability:

Scalability proved to be one challenge since the project sought to develop solutions implementable by actual grid operators for possibly millions of nodes, noted on PAGE6. Although having a dataset of 60,000 instances was quite large, scaling the system up to cope with larger, more intricate grid networks posed problems, a drawback further highlighted under literature review on PAGE7. This problem hindered the system's capacity to support increasing volumes of data and network sizes, typical in contemporary smart grids. PySpark's distributed computing paradigm was employed to ensure scalability by means of work distribution (PAGE4), but scaling the system to the size needed to continue operating and to be efficient at a larger scale needed better optimization. The report refers to this as an area for improvement to respond to requirements for real-world deployment in colossal grid infrastructures.

7. Interpretability vs. Performance Trade-off:

Model performance vs. interpretability raised the issue of trade-off, as addressed in PAGE35 model comparison. Models such as Random Forest provided insightful interpretability through feature importance scores (PAGE18), enabling understanding of what variables led to stability prediction, whereas the more performing models such as SVM (99.86% accuracy) and Deep Learning (99.50% accuracy) were less interpretable and acted as "black boxes'.This transparency can hamper trust and adoption by grid operators who require actionable knowledge to facilitate more informed decision-making. The authors experimented with different models in an attempt to achieve this trade-off, eventually choosing SVM due to its higher performance, but the trade-off in loss of interpretability remained. This case represents an endemic issue with AI-enabled systems, where optimizations for best accuracy are at the cost of explainability and must be interrogated over their use in practice.

9. CONCLUSION AND FUTURE SCOPE:

Conclusion:

The "Grid Stability Prediction" study is an ideal example of the capability of a data-driven, AI-powered solution to improve the prediction of the stability of contemporary electric power grids. With the help of PySpark for Big Data management and comparing four machine learning approaches Support Vector Machines (SVM), Random Forest (RF), Multi-Layer Perceptron (MLP), and Deep Learning (DL) with Keras the study was able to achieve impressive results, where SVM performed the best at 99.86% accuracy. This good precision, and well-balanced precision, recall, and F1-scores, demonstrates the effectiveness of the developed approach in predicting stable and unstable grid conditions from a data set of 60,000 examples from the University of California Machine Learning Repository (PAGE9). The addition of preprocessing techniques like SMOTE-ENN for the management of class imbalance (PAGE28) and 3-fold cross-validation to make the model stable (PAGE12) further improved the system's reliability over overcoming issues with data imbalance and overfitting. The gap between the typical rule-based grid monitoring and high-end AI-based solutions as elucidated in its objectives (PAGE5) has been covered by the project through offering a scalable and accurate framework that goes beyond conventional deterministic practices. This project, done under the guidance of Dr. V. Lakshmi Chetana during the academic year 2024-2025, provides a robust framework for real-time grid stability monitoring, which contributes immensely to power system analytics by enabling energy companies to prevent blackouts and ensure a continuous power supply (PAGE2). Lastly, the excellent performance of SVM and the combined comparative study (PAGE34) affirm the potential of machine learning to revolutionize the grid's management, paving the way for smart, dependable power infrastructures.

Future Scope:

The project establishes a sound foundation for forecasting grid stability, but some avenues of development in the future are born out of its findings and limitations, identified in the goals (PAGE6) and abstract (PAGE2). Among these is the validation of Deep Reinforcement Learning (DRL), which would arguably introduce real-time adaptability to the system, removing the liability of delay in prediction and building dynamic feedback against grid perturbations.This solution would complement the current Deep Learning model (99.50% accuracy) with continuous learning and decision-making capabilities, so it would be even more deployable in operational utilization in smart grids. Another way forward is the development of hybrid models by combining the strengths of statistical methods, machine learning, and real-time control systems, as suggested on PAGE6. These hybrids would deliver enhanced predictive accuracy by blending domain knowledge of the power system with AI-driven insight, potentially beating the current state of the art delivered by SVM. The scalability issue (PAGE6) must also be studied further, with follow-on work aimed at fine-tuning the system to handle larger data sets and larger grid networks, possibly with more advanced distributed computing approaches than PySpark. In addition, increasing the interpretability of the model is a crucial future goal, especially for highly performing yet black-box models like SVM and Deep Learning (PAGE35); techniques like SHAP (SHapley Additive exPlanations) or feature attribution can potentially render predictions more transparent to grid operators. Lastly, including the dataset to account for various real-world conditions like extreme weather or two-way energy transfer from renewables is going to improve the system's robustness and generalizability, filling gaps identified in the literature review. All of these are going to further enhance the project's impact, bringing it closer to an implementable solution for future power grid operation that ensures energy security and resilience in an increasingly complex energy environment.

References:

[1] W. Ahmed et al., "Machine Learning Based Energy Management Model for Smart Grid and Renewable Energy Districts," in IEEE Access, vol. 8, pp. 185059-185078, 2020,

doi: 10.1109/ACCESS.2020.3029943

[2] R. H. Byrne, T. A. Nguyen, D. A. Copp, B. R. Chalamala and I. Gyuk, "Energy Management and Optimization Methods for Grid Energy Storage Systems," in IEEE Access, vol. 6, pp. 13231-13260, 2018,

doi: 10.1109/ACCESS.2017.2741578

[3] Panagiotis D. Diamantoulakis, Vasileios M. Kapinas, George K. Karagiannidis,

Big Data Analytics for Dynamic Energy Management in Smart Grids, Big Data Research, Volume 2, Issue 3, 2015, Pages 94-101, ISSN 2214-5796,

Link: https://doi.org/10.1016/j.bdr.2015.03.003

[4] E. Hossain, I. Khan, F. Un-Noor, S. S. Sikander and M. S. H. Sunny, "Application of Big Data and Machine Learning in Smart Grid, and Associated Security Concerns: A Review," in IEEE Access, vol. 7, pp. 13960-13988, 2019

doi: 10.1109/ACCESS.2019.2894819

[5] Franović, B.; Baressi Šegota, S.; Anđelić, N.; Car, Z. Decentralized Smart Grid Stability Modeling with Machine Learning. *Energies* **2023**, *16*, 7562.

Link: https://www.mdpi.com/1996-1073/16/22/7562

[6] Lahon, P.; Kandali, A.B.; Barman, U.; Konwar, R.J.; Saha, D.; Saikia, M.J. Deep Neural Network-Based Smart Grid Stability Analysis: Enhancing Grid Resilience and Performance. *Energies* **2024**, *17*, 2642.

Link: https://www.mdpi.com/1996-1073/17/11/2642

[7] A. M. Jadhav and N. R. Patne, "Priority-Based Energy Scheduling in a Smart Distributed Network With Multiple Microgrids," in IEEE Transactions on Industrial Informatics, vol. 13, no. 6, pp. 3134-3143, Dec. 2017

doi: 10.1109/TII.2017.2671923

[8] Qin Wang, Wei Yao, Jiakun Fang, Xiaomeng Ai, Jinyu Wen, Xiaobo Yang, Hailian Xie, Xing Huang, Dynamic modeling and small signal stability analysis of distributed photovoltaic grid-connected system with large scale of panel level DC optimizers, Applied Energy, Volume 259, 2020, 114132, ISSN 0306-2619

Link: https://doi.org/10.1016/j.apenergy.2019.114132

[9] Matheus Sabino Viana, Giovanni Manassero, Miguel E.M. Udaeta, Analysis of demand response and photovoltaic distributed generation as resources for power utility planning, Applied Energy, Volume 217, 2018, Pages 456-466, ISSN 0306-2619,

Link: https://doi.org/10.1016/j.apenergy.2018.02.153.

[10] Rolf Golombek, Arne Lind, Hans-Kristian Ringkjøb, Pernille Seljom, The role of transmission and energy storage in European decarbonization towards 2050, Energy, Volume 239, Part C, 2022, 122159, ISSN 0360-5442,

Link: https://doi.org/10.1016/j.energy.2021.122159.

[11] Hans-Kristian Ringkjøb, Peter M. Haugan, Pernille Seljom, Arne Lind, Fabian Wagner, Sennai Mesfun, Short-term solar and wind variability in long-term energy system models - A European case study, Energy, Volume 209, 2020, 118377, ISSN 0360-5442,

Link: https://doi.org/10.1016/j.energy.2020.118377.

[12] Aune, F. R., & Golombek, R. (2021). Are Carbon Prices Redundant in the 2030 EU Climate and Energy Policy Package? The Energy Journal, 42(3), 225-264.

Link: https://doi.org/10.5547/01956574.42.3